REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-05-

0375

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
15 August 2005	Final Technical Report	10/15/2004 - 7/15/2005

4. TITLE AND SUBTITLE
(STTR PHI) Optimal Training System

5a. CONTRACT NUMBER
FA9550-05-C-0005

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
N/A

6. AUTHOR(S)
Dr. Brad Best
Dr. Marsha Lovett

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
N/A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Micro Analysis and Design, Inc.
4949 Pearl East Circle
Suite 300
Boulder, CO 80301

8. PERFORMING ORGANIZATION REPORT NUMBER MBO0003Z

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
USAF, AFRL
AF Office of Scientific Research
4015 Wilson Blvd Rm 713
Arlington VA 22203-1954

NL

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR/NL
Dr. Robert SORKIN

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public release; distribution unlimited

13. SUPPLEMENTARY NOTES
NONE

14. ABSTRACT

Successful training in complex environments is normally accomplished through the interaction of a trainee and a skilled expert, but experts are an expensive commodity. Using an optimal model of task performance subject to human constraints may be a more efficient way to develop models of skilled human performance for use in training, especially since optimal models are simpler to validate, test, and debug than corresponding expert models. In addition, constrained optimal models can be constructed in domains where no experts are available or even exist. Using a simulated task environment (STE) permits the necessary close model-trainee interaction by enabling the construction of optimal performance models that perform the same task as the trainee using the same interface while closely observing and guiding trainee performance. We have developed a methodology for using a normatively correct task model as the core engine of an automated tutor for a national missile defense (NMD) task STE. This methodology has allowed us to explore: 1) the relative impact of expert versus optimal feedback, 2) the locus of learning within the NMD task, 3) the differential impact of providing feedback on strategy selection, and 4) methodologies for constructing tutors directly from expert performance data.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT UNCLASSIFIED	b. ABSTRACT UNCLASSIFIED	c. THIS PAGE UNCLASSIFIED	NONE	67	Dr. Brad Best
					19b. TELEPHONE NUMBER *(include area code)* 303 442-6947 x177

Contractor's Progress, Status, and Management Report

for the

Optimal Training Systems STTR

Contract Order Number: FA9550-05-C-0005

Period Covered:

Date of Preparation: 8/15/05

Prepared By:

Micro Analysis & Design
Bradley J. Best
4949 Pearl E. Circle, Suite 300
Boulder, CO 80301

Carnegie Mellon University
Marsha Lovett
Cyert Hall, Suite 120
5000 Forbes Avenue
Pittsburgh PA 15213

Prepared For:

Air Force Office of Scientific Research
Dr. Robert D. Sorkin
AFOSR/NL
4015 Wilson Blvd., Rm. 713
Arlington, Va 22203-1954

UNCLASSIFIED

Table of Contents

Table of Contents 2
Abstract 3
Anticipated Benefits and Potential Commercial Applications 3
Introduction 4
Overview of Completed Research 5
Lessons Learned about the NMD Task Structure from Pilot Studies 6
  Human Performance 6
  Preliminary Modeling Findings 6
  Redesigning the NMD Task 7
Methodology for Developing a Training Tutor Using an Ideal Cognitive Model 7
    Capturing individual decision-making with instance-based modeling 8
Human Performance on the NMD Feedback Task 10
  The National Missile Defense (NMD) Task 10
  Comparison among Feedback Conditions 11
  Learning 17
  Individual Differences 19
Modeling the NMD Task 20
  Models Developed for the Project 21
    Pure Optimal Model 21
    Rule-based Cognitive Model 21
    Instance-based Near Optimal Model 22
  Relating Model Performance to Human Performance 25
    Learning in the NMD Model 27
  Representational Choices 29
  Optimal Feedback versus Expert Feedback 29
  Instant Feedback versus Delayed Feedback 30
General Discussion and Conclusions 30
Future Directions 30
Future Technical Objectives 32
Commercialization 39
Appendices 42
  Appendix 1: Optimal Model 42
  Appendix 2: Expert (Cognitive) Model 45
  Appendix 3: Model Trace 56
  Appendix 4: Experiment Instructions 67

Abstract

Successful training in complex environments is normally accomplished through the interaction of a trainee and a skilled expert, but experts are an expensive commodity. Using an optimal model of task performance subject to human constraints may be a more efficient way to develop models of skilled human performance for use in training, especially since optimal models are simpler to validate, test, and debug than corresponding expert models. In addition, constrained optimal models can be constructed in domains where no experts are available or even exist. Using a simulated task environment (STE) permits the necessary close model-trainee interaction by enabling the construction of optimal performance models that perform the same task as the trainee using the same interface while closely observing and guiding trainee performance. We have developed a methodology for using a normatively correct task model as the core engine of an automated tutor for a national missile defense (NMD) task STE. This methodology has allowed us to explore: 1) the relative impact of expert versus optimal feedback, 2) the locus of learning within the NMD task, 3) the differential impact of providing feedback on strategy selection, and 4) methodologies for constructing tutors directly from expert performance data.

Anticipated Benefits and Potential Commercial Applications

The completion of Phase I of this project has produced a cognitive modeling methodology for developing human behavior representation systems that has the potential to avoid many of the shortcomings of the knowledge engineering approach to authoring those systems. The approach detailed in this report yields realistic and accurate models of normative and expert performance while minimizing the investment that must be made to produce and maintain those models.

This architecture has the potential to form an ideal core for developing automated training systems for the military, teaching humans to perform complex tasks and assessing their performance in sustained operations. Example applications include piloting an unmanned aerial vehicle, operating an emerging weapon system, and allocating ground based interceptors in a missile defense environment.

Beyond the training domain, this approach could be used in any application requiring accurate, affordable human behavior representation. For example, it could be used to implement autonomous robot navigation in an interior space. It could form the core of an intelligent assistant in a decision support system. It could be used in human-computer interface design. It could be used to create a model of "typical" (vs. expert) human performance, which could then be used to represent a believable human entity for Computer Generated Force (CGF) experiments.

The most obvious commercial products to come from this effort would be more effective automated training systems, constructed with much less time and money than is currently typical. The ability to accurately model expert and normative performance could transform something like the Microsoft Paper Clip from an annoying distraction into a truly useful assistant. This research could even have an impact on the development of computer generated avatars and non-player characters for the computer gaming industry.

The results of this research are also applicable, as has already been demonstrated for the Traveling Salesperson Problem, to solving problems which are intractable using traditional computational methods, but which become quite tractable when the problem representation exploits the kinds of representations and processes used by humans in solving the problem.

Introduction

We have developed the concept of using a normatively correct model of task performance as the core engine of an automated tutor with an initial application to a national missile defense (NMD) task STE. The NMD STE is a complex task requiring skilled operators to allocate assets under time constraints to minimize expected losses, yet is amenable to construction of an optimal model, and therefore allows us to explore a normative modeling-based tutoring approach. In this task, trainees allocated some number of ground based interceptors (GBIs) to protect five cities from a likely nuclear attack. The four main technical objectives for Phase I of this research are as follows:

1. Investigate the different consequences of a normative vs. an expert model. These two approaches may lead to different performance of the model itself, and as the basis for a training system. That is, an expert model may or may not perform the task as well as an optimal system, and an optimal system may or may not direct a training session as well as an expert model. This makes it important to determine the ways in which the two approaches differ and whether these differences offer different advantages or disadvantages for tutoring. We will conduct this analysis in the context of the NMD task and attempt to extend this understanding to other domains. (The distinction between normative and (human) expert performance is an important, general issue in training, developing tutoring systems, and in understanding expertise. What we find here will likely transfer to training issues in other domains.)
2. Build on an existing model of normative performance in NMD task to incorporate results from recent research on decision-making biases (e.g., framing effects). This will allow us to explore whether optimal training can result simply from guidance in optimal behavior, or whether add-on of refinements based on current research can enhance training. The NMD task is particularly appropriate for this purpose because framing effects are known to occur in this task, and the augmentation of an optimal model with the intention of countering these biases would be straightforward. That is, training with an optimal model may result in human performance that is a product of optimal behavior and their existing biases. It may be possible to augment the optimal model in a way that results in the reduction or elimination of their existing biases beyond that which can be accomplished using a pure optimal model.
3. Develop a model of (relatively) expert human performance based on empirical data. This model will be a process model of performance that also offers an interesting comparison to the normative and augmented normative models discussed above. For example, by using this model it would be possible to determine if there is a continuum from normative to expert to novice performance or whether there are qualitative differences between the categories. The effect on training of optimal vs. expert model will be studied by switching the models while keeping the rest of the training apparatus constant, which will

provide empirical evidence as to whether better performance is preferable in a tutoring model to human-like processing.

4. Investigate the cognitive limitations of human performance and the associated individual difference variables. It is possible that human performance is effectively optimal given the constraints imposed by the perceptual system, memory system, etc. By carefully comparing human performance with the various models proposed above, it should be possible to isolate the factors that influence human performance on the task. In the human performance model, one could then investigate how variations in the key individual difference variables (such as working memory capacity, perceptual speed) influence overall performance. This will extend the work done by Lovett, Reder, and Lebiere (1999)[1]. One can also evaluate the impact of individual differences on learning, such as speed of learning and choice of strategies.

Overview of Completed Research

Phase I of this research focused on using an optimal algorithm to design a training tutor for a complex task. In this task, a national missile defense (NMD) task, trainees allocated some number of ground based interceptors (GBIs) to protect five cities from a likely nuclear attack. The task, which requires balancing the probabilistic expectations of lives saved and lost with the possibility of future attack, is a difficult task for students to master. In fact, students using the interface in the absence of a tutoring program are still learning after several days of practice and training, and have difficulty in applying the mathematical model of expected lives saved to the real-time problem of GBI allocation. Part of this difficulty appears to be that, though the optimal model is straightforward, there is no easy way for students to induce that model while performing the task – they are busy completing the task and the workload precludes them from deducing a better way to do the task. This forces them to apply a more general heuristic, which appears in this case to be a memory-based matching process. However, the memory they have for various configurations of city populations, GBIs, and the resulting outcomes are, due to the fact that they are learning the task, rife with errors and suboptimal decisions. That is, these students are forced to bootstrap their learning onto their own initial guesses and are left attempting to improve on that performance. This choice of a training domain, as a result, is an ideal domain for constructing a training tutor to help students learn the correct strategy, and has led us to develop a method for constructing a training tutor based on the optimal model of problem performance.

Our first attempts involved training mathematics experts to directly implement the optimal strategy. However, many of the steps in this strategy, though known by the expert, were simply too demanding in terms of time required, working memory demands, etc., to be carried out in the time allotted during the NMD task. We realized that instead we needed to express the optimal algorithm in a way that was computable by people within the time allowed for task completion. Our preliminary findings indicate that a viable method of constructing a training tutor is to encode the optimal algorithm within the framework of a cognitive architecture. This leads straightforwardly to the imposition of the constraints of the cognitive architecture (i.e., human

[1] Lovett, M. C., Reder, L. M., & Lebiere, C. (1999). *Modeling working memory in a unified architecture: An ACT-R perspective.* In A. Miyake & P. Shah (Eds.) Models of Working Memory. pp. 135-182. Cambridge, MA: Cambridge.

limitations) on the optimal algorithm, which in turn produces a near optimal algorithm that a trainee is capable of performing. The task then becomes training the student in that near optimal strategy. The primary innovation here is the use of the limitations encoded in a cognitive architecture to prevent, or at least make extremely difficult, the expression of an algorithm that exceeds the information processing capabilities of the trainee. That is, if a process or algorithm can be written in the language of a cognitive architecture, it can be learned and performed by a human trainee. This approach represents a novel departure from the traditional approaches in automated training. As such, the methodology developed in the course of this research bears further description, and will be presented below along with the other results of the Phase I research.

Lessons Learned about the NMD Task Structure from Pilot Studies

Human Performance

The NMD task, which requires balancing the probabilistic expectations of lives saved and lost with the possibility of future attack, is a difficult task for students to master. In fact, students using the interface in the absence of a tutoring program are still learning after several days of practice and training, and have difficulty in applying the mathematical model of expected lives saved to the real-time problem of GBI allocation. Part of this difficulty appears to be that, though the optimal model is straightforward, there is no easy way for students to induce that model while performing the task – they are busy completing the task and the workload precludes them from deducing a better way to do the task. This forces them to apply a more general heuristic, which appears in this case to be a memory-based matching process. However, the memory they have for various configurations of city populations, GBIs, and the resulting outcomes are, due to the fact that they are learning the task, rife with errors and suboptimal decisions. That is, these students are forced to bootstrap their learning onto their own initial guesses and are left attempting to improve on that performance. This choice of a training domain, as a result, is an ideal domain for constructing a training tutor to help students learn the correct strategy, and has led us to develop a method for constructing a training tutor based on the optimal model of problem performance.

Preliminary Modeling Findings

Our first attempts involved training mathematics experts to directly implement the optimal strategy. However, many of the steps in this strategy, though known by the expert, were simply too demanding in terms of time required, working memory demands, etc., to be carried out in the time allotted during the NMD task. We realized that instead we needed to express the optimal algorithm in a way that was computable by people within the time allowed for task completion. Our preliminary findings indicate that a viable method of constructing a training tutor is to encode the optimal algorithm within the framework of a cognitive architecture. This leads straightforwardly to the imposition of the constraints of the cognitive architecture (i.e., human limitations) on the optimal algorithm, which in turn produces a near optimal algorithm that a trainee is capable of performing. The task then becomes training the student in that near optimal strategy. The primary innovation here is the use of the limitations encoded in a cognitive architecture to prevent, or at least make extremely difficult, the expression of an algorithm that

exceeds the information processing capabilities of the trainee. That is, if a process or algorithm can be written in the language of a cognitive architecture, it can be learned and performed by a human trainee.

Redesigning the NMD Task

Our pilot studies had demonstrated that the NMD task in the form it has been studied is fundamentally intractable from a human cognitive perspective. The task interface prevents making progress on the problem until only a small amount of time is available to complete it, and this remaining time is insufficient for any amount of cognitive activity that can even start to approximate a good solution. This forces participants into a rapid decision process which, in the absence of a body of experience from which to draw, forces participants into generally inferior decisions and hampers learning.

As a result of these difficulties, we have extended the NMD task with two goals in mind: 1) provide a task environment where a high level of performance is, at least in theory, achievable, and 2) provide feedback in several forms that could be reasonably expected to impact learning in the task. This feedback comes in two primary forms: immediate, stepwise feedback, and delayed, holistic feedback. The key distinction between these is that immediate feedback can be expected to impact the solution process, while delayed feedback is more inclusive of a variety of strategies.

Methodology for Developing a Training Tutor Using an Ideal Cognitive Model

We have identified two primary ways to develop a training tutor based on an ideal or optimal model. The first of these methods, the generation of a near-optimal model from an optimal model, involved producing an ACT-R cognitive model of task performance from the algorithmic specification of the optimal model. The ACT-R architecture imposes a set of human constraints on processing that produce performances that are less than optimal (i.e., not perfect), but which are, as a result, learnable. The second method, the induction of an expert model from a trace of expert human performance, involved producing an ACT-R cognitive model of task performance using the learning aspects of the architecture to imitate the observed actions of a human expert.

The model induction approach leverages instance-based learning techniques based on the ACT-R cognitive architecture[2] to model the underlying decision processes that operate on problem abstractions having ideal (optimal) solutions. Problems with ideal solutions provide a unique environment for studying human decision-making, since the decisions can be situated in the context of ideal performance, making it possible to isolate both areas where human performance is lacking, and areas where human performance challenges or exceeds the best artificial intelligence approaches.

One example problem domain where this technique has been applied is the Traveling Salesperson Problem. The Traveling Salesperson Problem (TSP) is a computationally hard (NP-complete) problem where time complexity of the best known solutions scales exponentially in proportion to the number of "cities" (points) that must be visited exactly once on a sales route.

[2] Anderson, J. R. & Lebiere, C. (1998). *The atomic components of thought.* Mahwah, NJ: Erlbaum.

Best (2004)[3] demonstrated that human solvers tackle a much simpler problem than the unconstrained TSP by using hierarchical clustering and goodness of path determination based on established cognitive principles to reduce the problem to one that can be solved in linear time in the number of cities. This finding was validated through the construction of a computational model that also predicted solutions that were linear in time given the cognitively constrained representation, and through a demonstration that human solvers did not need or use non-local high-frequency spatial information to solve the problem. The key to capturing human performance in this task was determining the global structure solvers used to organize the problem-solving episode, and uniting this with a local process that captured individual decisions.

Capturing individual decision-making with instance-based modeling

Instance-based models of human performance have been constructed for many individual decision processes. For example, Lebiere, Wallach, and West (2000)[4] showed how the fundamental memory processes encoded in the ACT-R cognitive architecture can account for human behavior in games such as the Prisoner's Dilemma, a game in which a payoff matrix specifies positive and negative payoffs based on not just the player's move, but the opponent's move as well (games like this are of particular interest to economists since many economic systems can be analyzed in game-theoretic terms). In this case, the human performance was captured through two simple uniform architectural processes – power law learning and decay, and stochasticity – which were incorporated into a general strategy based on determining the most likely outcome given each of the player's potential moves, and choosing that move with the best of the likely outcomes (stochasticity is essential to prevent the opponent from making easy predictions of the player's strategy). The ACT-R equation that produced both adaptation and stochastic behavior for this model is given below:

$$A = \ln \sum_{j=1}^{n} t_j^{-d} + N(0, \frac{\pi \bullet s}{\sqrt{3}}) \cong \ln \frac{n \bullet L^{-d}}{1-d} + N(0, \frac{\pi \bullet s}{\sqrt{3}})$$

The first term of this sum represents the strengthening in memory of a particular piece of information (a chunk) each time it is either retrieved from memory or (re)created. In this term, t_j represents the time since the j^{th} reference, while n is the number of references to the chunk, and d is the decay rate. The implications of this formula are that activation increases with use and decreases with time with a functional form that produces both the power law of learning and the power law of decay. Assuming even distribution of the chunk references over time allows for approximating activation with the shown function of decay, number of chunks, and total life of the chunk, L. Stochasticity is provided by the second term of the sum, normally distributed noise with a mean of 0 and a standard deviation determined by the activation noise parameter, s. These relatively simple equations provide the basis for instance-based learning and decision-making within the ACT-R architecture.

---

[3] Best, B. J. (2004). *Modeling Human Performance on the Traveling Salesperson Problem: Empirical Studies and Computational Simulations*. Doctoral dissertation, Department of Psychology, Carnegie Mellon University, Pittsburgh, PA.

[4] Lebiere, C., Wallach, D., & West, R. L. (2000). A memory-based account of the prisoner's dilemma and other 2x2 games. In *Proceedings of International Conference on Cognitive Modeling*, 185-193. NL: Universal Press.

Our work on the NASA Human Performance Modeling project (see related work section) used an instance-based decision process within an ACT-R model of a pilot's behavior during an approach and landing task. In this case, the flight control inputs were determined by the pilot model by retrieving prior behavior from memory (e.g., given a certain altitude and airspeed, recall the appropriate setting for the flaps and adjust the controls to that setting if they are different). By seeding the system with several instances of typical pilot behavior (correct exemplars), the model was able to behave across a range of situations it had never seen before. In general, instance-based techniques such as the method described here naturally generalize to situations which are similar to those that have been seen before, but are not exact matches to previous situations or behavior.

Gunzelman, Anderson, and Douglass (submitted)[5] have used the instance-based approach to account for a basic decision process in a model of human behavior in a spatial localization task within the ACT-R framework. The decision to be made is the selection of a point on a two-dimensional view of a scene (a view of a circular field containing an arrangement of identical conical objects from above) that corresponds to the viewpoint shown in a three-dimensional view of the scene (a viewer-centered perspective from an arbitrary location on the perimeter of the field facing the center of the field). If the chosen point falls within fifteen degrees of the actual location, the response is coded as correct. Otherwise, the trainee receives feedback indicating an incorrect solution.

The modeling approach taken by Gunzelman et. al. was to have the model choose whether to respond at a particular moment in time based on the information accumulated at that point and memory for success and failure of previous decisions given varying amounts of information. Although the model will often make correct decisions (hits), occasionally, this decision model will also make errors by responding positively in the absence of enough evidence (false alarms). This model may also choose not to respond when it has sufficient evidence (misses), or choose not to respond where there is not sufficient evidence (correct rejections).

This can be seen as an encoding of a basic signal-detection process at the output stage of a cognitive decision process. More information increases discriminability, while the success and failure of previous similar decisions biases the system towards either responding or seeking more information. Using this decision framework, bias and discriminability can be manipulated through modifying architectural parameters. This allows for straightforward modeling of individual differences on a variety of tasks (and, in fact, potentially for the same individual across tasks) and the study of the impact of these individual differences on the acquisition of the skill in question.

---

[5] Gunzelmann, G., Anderson, J. R., & Douglass, S. (submitted). Orientation tasks with multiple views of space: Strategies and performance. *Spatial Cognition and Computation.*

Human Performance on the NMD Feedback Task

The National Missile Defense (NMD) Task

The version of the NMD task that is being discussed in this report is primarily the augmented NMD task rather than the original version. However, the inspiration for the augmented version came directly from the preliminary modeling studies, so this section will discuss both versions of the task.

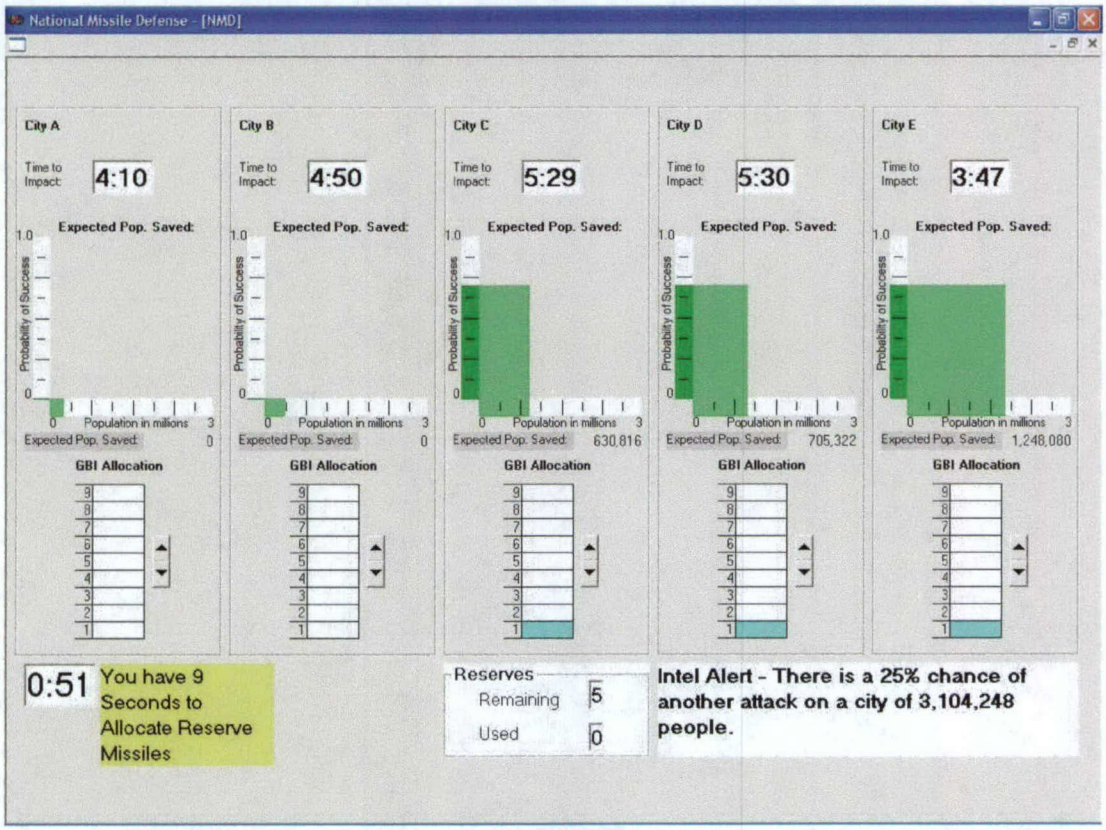

Figure 1. The National Missile Defense (NMD) Task Interface

The screen shot above shows the primary NMD task display, which consists of a panel containing a set of cities, each of which has a population and some assignment of ground based interceptor (GBI) missiles to defend it from possible attack. The center bottom of the display shows the number of GBIs used and remaining in reserves while the lower left portion of the screen is focused on a countdown. The number of GBIs used is represented using a color bar and a slider control. The population and percentage of savings of life can be read off of the area for each city, most easily by paying attention to the area produced by combining these measures. The green areas above, then, represent the lives that will be saved by allocating the missiles as chosen above.

In addition to this screen, every trial gives one of three forms of feedback: 1) a summary of what the student chose to do, 2) an after-action review of the outcome of the actions taken by the

student and a comparison to optimal allocations, and 3) stepwise feedback during problem solution in the form of beeps when the student strays off of the optimal solution path. The screen shot below shows a screen from the elaborative feedback condition:

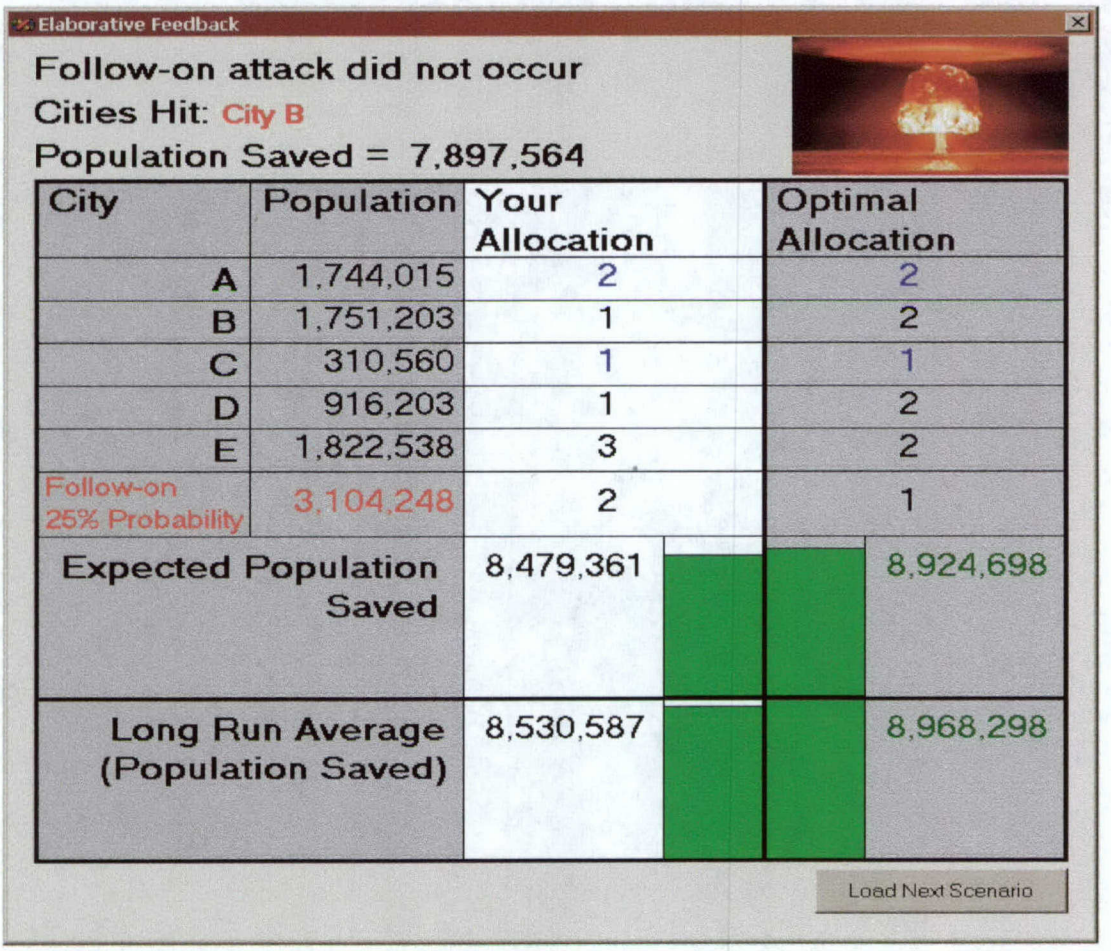

Figure 2. NMD Task Feedback Screen

Comparison among Feedback Conditions

This section describes the results of a study of human performance in the NMD task under various feedback conditions. The goal is to describe the nature of performance relative to optimal and to evaluate which of the training conditions tested was most effective.

The method was straightforward. Participants were recruited to complete the NMD task and several others for pay. For the NMD task, they read a powerpoint file with instructions (see Appendix 4 for the instructions), saw a brief demonstration of the task interface and began working through the NMD scenarios (also referred to as trials, below). Each scenario was drawn at random from a set of possible scenarios with 5 different city populations, a follow-on probability of attack on a 6th city (either .25 or .75) and a population for the 6th city. Participants completed 72 trials in two blocks of 36 trials each.

Recall that the optimal algorithm for this task can be defined in terms of a sequence of moves (i.e., GBI allocations), for which the next move taken is always the one that produces the largest increase in the expected value of lives saved (or, similarly in the "lives lost" frame, the largest decrease in the expected value of lives lost). With this specification of the optimal strategy, each step can be defined as on or off the optimal solution path. Moreover, this allows the application of an immediate feedback strategy like that used in cognitive tutors (Anderson, Corbett, Koedinger, & Pelletier, 1995)[6], namely, where students' problem-solving steps are communicated to the intelligent tutoring system through a computer interface and the intelligent tutor interrupts with feedback only when their step is off a known solution path. In our case of training participants to produce an optimal solution in the NMD task, we have implemented a version of this feedback scheme that we call "stepwise feedback." It is just like the immediate feedback approach used in cognitive tutors with the following two exceptions: first, participants are interrupted when their step is off the *optimal* path (not just off a known solution path), and second, the nature of the feedback is a simple beep, indicating to participants that their move was in error. While previous research has shown that highly refined feedback messages can lead to improved learning outcomes (over that attained with more simple feedback), there are no clear principles to guide the design of such feedback messages that would apply to the NMD task. Moreover, our goal in developing the stepwise feedback training condition was to evaluate whether simply providing some minimal, immediate feedback regarding whether each step was on the optimal path would produce improved learning gains. Note that the NMD software was adapted for this training condition so that each time a participant locked in a particular increase or decrease in GBI allocation to a particular city, the software would compare this re-allocation with the next move in the optimal solution. If the participant's move did not match the next optimal move, the software would produce an immediate beep. Otherwise, the participants would be left to continue their work without interruption.

To evaluate the stepwise feedback condition, we compared its effectiveness along several measures to the effectiveness of the two other conditions— outcome feedback and summary feedback. In the "outcome" condition, participants are shown, after each trial, a table aligning the GBI allocations they chose for each city with the GBI allocations of the optimal solution as well as expected values of lives saved/lost. In the "summary" condition, participants received no feedback after each trial and only at the end of each set of trials were shown more generically how well they had performed. Note that the outcome condition is somewhat similar to conditions that have been used in other research on the NMD, in which participants received feedback on their *solution* after each trial. The key distinction between the outcome and the stepwise feedback conditions is whether the feedback involves the solution and comes at the end of each trial (outcome condition) or each step toward a solution and comes continually within each trial (stepwise condition).

One measure for comparison among the three conditions is the distance from optimum, calculated as the difference in expected value of lives saved/lost for the optimal GBI allocation versus the participants' GBI allocation (averaged across all trials). Figure 3 shows a boxplot of these averages by condition. An ANOVA reveals there is a significant difference among the three groups, $F(2, 17) = 5.68$, $p < .05$). As visual inspection of the medians suggests, there is a

[6] Anderson, J. R., Corbett, A. T., Koedinger, K., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of Learning Sciences*, 4, 167-207.

significant difference between the summary condition and the other two ($p < .05$ for both) but not a significant difference between the outcome and stepwise feedback conditions. It is worth noting that the stepwise feedback condition shows much greater variability in this distance-from-optimal measure, with a standard deviation of 57,725 and a range of 159,972 – 78 = 159,894 whereas the outcome condition had a standard deviation of 25,693 and a range of 101,000 – 30,852 = 70,148.

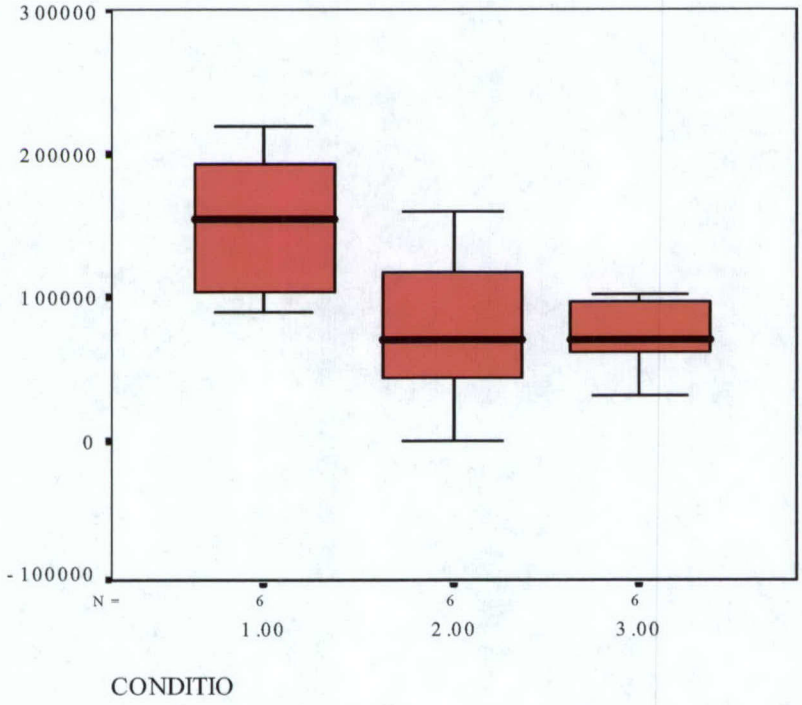

Figure 3. Differences among conditions in the average distance from optimal solution. Condition 1=Summary; Condition 2=Stepwise, Condition 3=Outcome.

It is possible, however, that the randomization of participants into conditions was not quite adequate and that, in particular, the stepwise feedback participants were for some unknown reason a slightly different sample from those in the other two conditions. If so, this could partly explain why the stepwise condition's performance was not significantly better than the outcome condition's. To begin to investigate this issue, trial 1 performance on this measure (difference from optimal solution) is plotted across the three groups in Figure 4. Although there is not a significant difference among the conditions, $F(2, 16) < 1$, the greater variability among the stepwise feedback condition is again evident, suggesting that indeed there may be some a priori differences among the groups. We will investigate this further when we address the question of individual differences (see section on individual differences below).

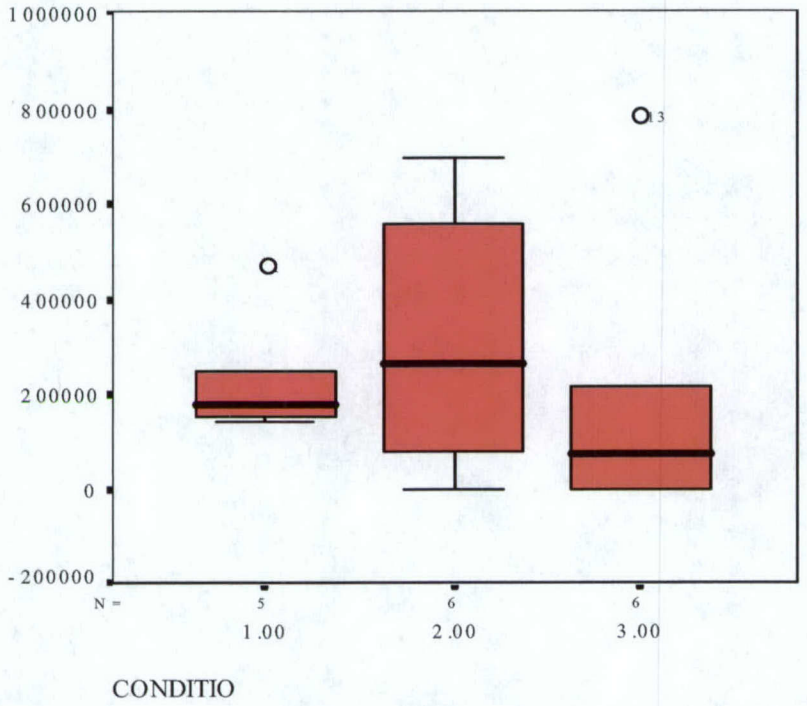

Figure 4. Distance from optimal solution for trial 1 across conditions. Condition 1=Summary; Condition 2=Stepwise, Condition 3=Outcome.

Another measure of the effectiveness of the conditions involves assessing how sensitive participants were to different factors that made individual scenarios difficult. We had two a priori hypotheses regarding what factors would make problems difficult. One was that scenarios where the optimal GBI allocation involved leaving at least one city with no GBIs would be particularly difficult. This hypothesis was based on previous research with the NMD task[7] and followed the idea that participants would not want to leave a city completely unprotected. The second hypothesis involved the number of GBIs that were to be held in reserve under the optimal solution. This hypothesis stemmed from the idea (also based on past work) that participants would focus more on the cities highlighted in the display and be both less aware of the consequences for the follow-on city and less able to perform the adequate computations for GBI allocation for the follow-on city. The second hypothesis was not supported: Measures of participant performance did not differ as a function of optimal number of GBIs to be allocated to the follow-on city. However, the first hypothesis was supported by the data in that participants' average distance from optimal was higher (i.e., worse) on scenarios where the optimal allocation left at least one city unprotected (i.e., with 0 GBIs allocated to it): 150,237 vs. 82,645.

This sensitivity to this difficulty factor, found over all participants on average, provides a means for comparing among the conditions because it allows us to test the degree to which the different training conditions enabled participants to avoid the "trap" of allocating at least some GBIs to all

[7] McDermott, P., Hutchins, S., Barnes, M., Koenecke, C., Gillan, D., & Rothrock, L. (2003). The presentation of risk and uncertainty in the context of national missile defense simulations. *Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting*. 13-17 October, Denver, CO.

cities rather than keeping in mind the criterion for optimal solution – maximizing the expected value of number of lives saved. In other words, one would consider training more effective if participants performed well regardless of this difficulty factor. Figure 5 shows the relevant data: participants' distance from optimal solution for difficult and easy scenarios, across the three conditions. The interaction in these data shows a similar pattern as the previous "distance from optimal" results, namely, that the stepwise condition and the outcome condition perform almost indistinguishably (both are relatively insensitive to scenario difficulty), while the summary condition performs worse (shows more sensitivity to scenario difficulty).

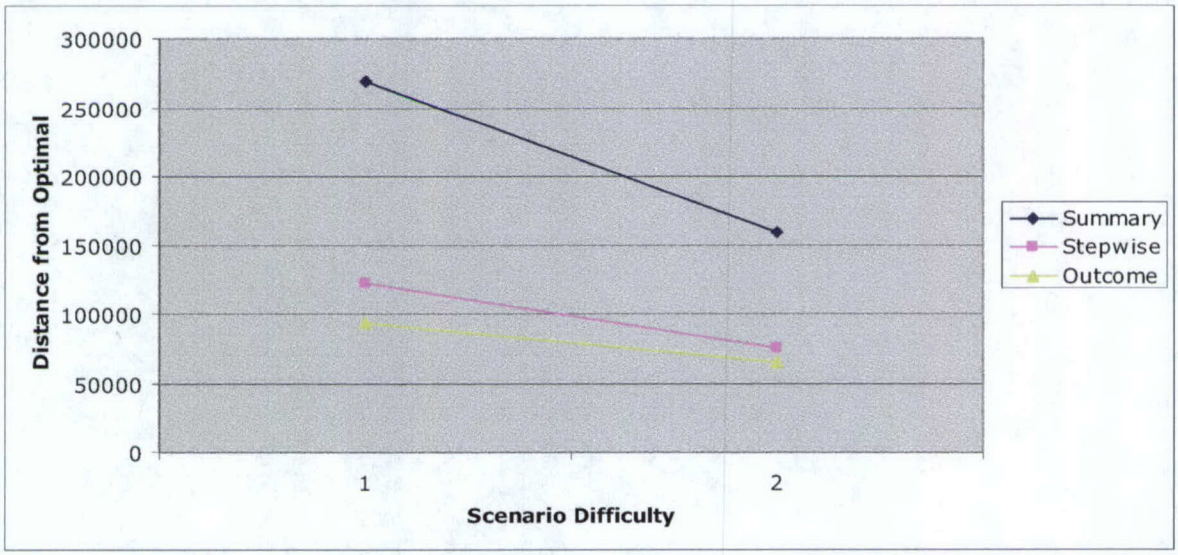

Figure 5. Sensitivity to difficulty factor (optimal solution involves leaving a city with 0 GBIs) across conditions. On the x-axis 1 represents "difficult" scenarios where the optimal solution did involve leaving a city with 0 GBIs, and 2 represents "easy" scenarios where the optimal solution did not involve leaving a city with 0 GBIs.

Finally, another measure along which we can compare across conditions is latency, namely, how long participants took on average to make their GBI allocations for each trial. Note that there was a maximum time allotted for GBI allocation and the nature of the task display invited the idea that time was critical (i.e., several clocks ticking down to impact time!), so finding any difference here would mean that the fastest group had acquired a better level of efficiency throughout their training.

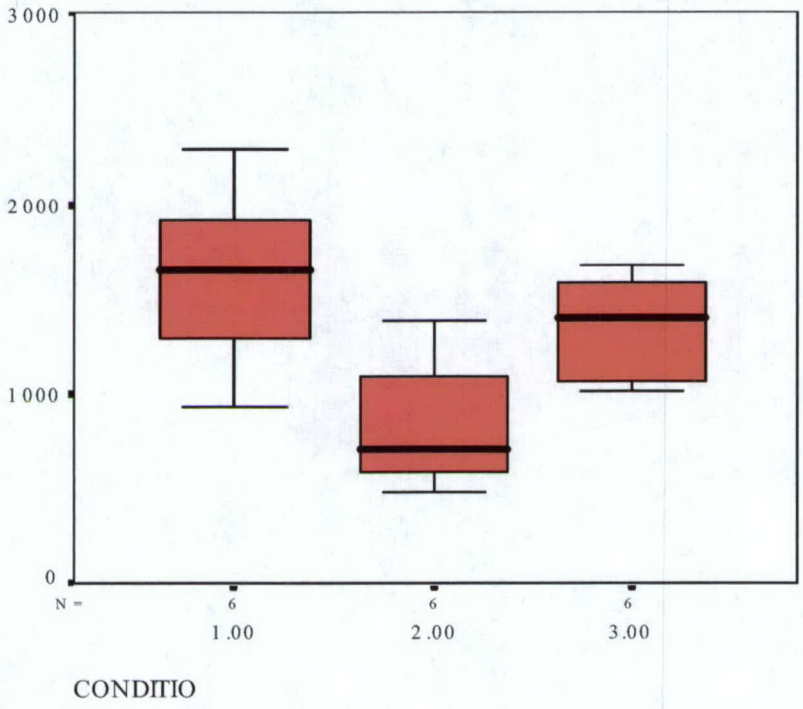

Figure 6. Average latency across conditions. Condition 1=Summary; Condition 2=Stepwise, Condition 3=Outcome.

Figure 6 shows the average latency across conditions. As this display suggests, there was a significant difference among conditions, $F(2, 17) = 6.59$, $p < .01$. In particular, post-hoc tests revealed that the effect stems from the stepwise condition being significantly faster than both of the other conditions ($p < .05$). Figure 7 shows this effect across time, emphasizing the fact that the stepwise group is able to make GBI allocations rather efficiently from the beginning and speeds up from there, whereas the other two groups start slower and speed up but do not quite reach as low a latency by the end of training.

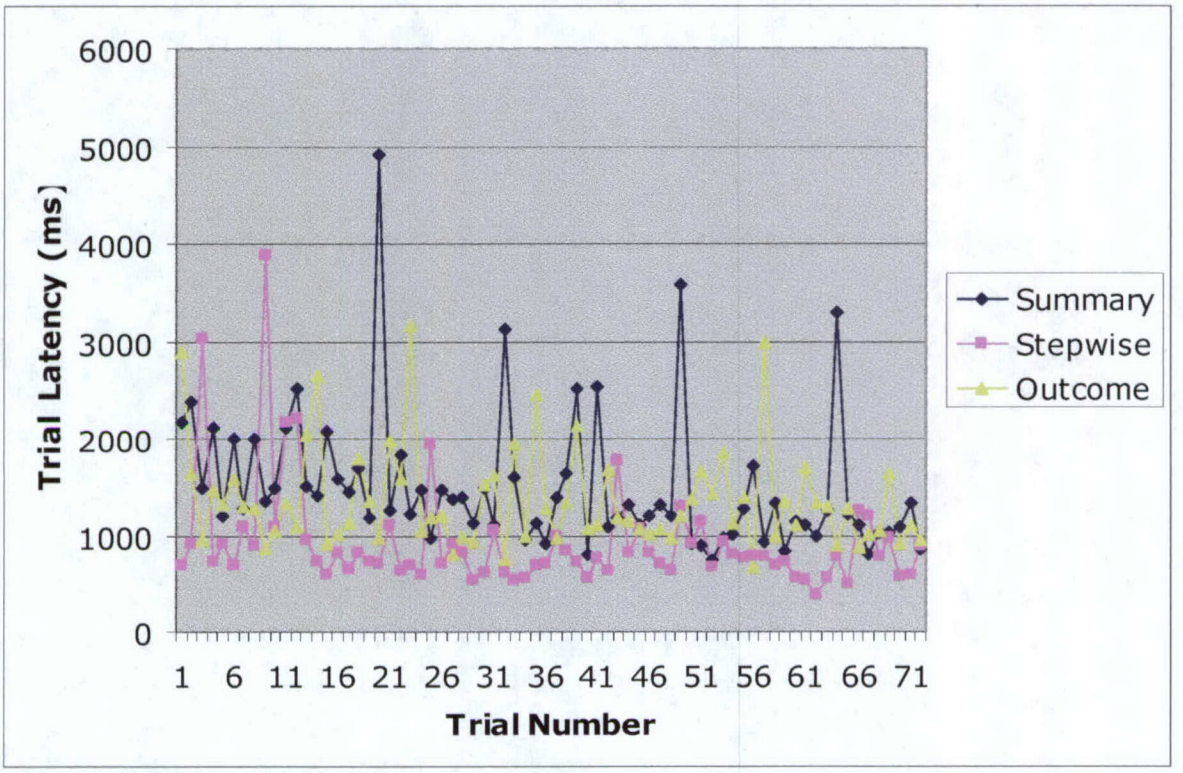

Figure 7. Latency across trials and across conditions

To summarize this section comparing the three conditions, it appears that both the outcome and stepwise feedback conditions were more effective than the summary condition when participants' ability to come close to the optimal solution was assessed and when this ability was assessed as a function of scenario difficulty. The outcome and stepwise feedback conditions, however, were not significantly different from each other in these two measures of effectiveness. They did diverge, however, when the measure of latency was explored. Here, the result was that the stepwise condition was faster (even compared to the outcome condition) and showed a significant speed-up over the course of training. The nature of the learning in the stepwise group and how this condition achieved greater efficiency (as measured via step latency) will be explored in the next subsection.

Learning

As discussed above, the stepwise feedback condition performed as well as the outcome condition but was able to achieve that performance level with faster GBI-allocation times. This kind of training advantage, equal performance with faster learning, is a relatively common result in the intelligent tutoring literature (e.g., Anderson, Conrad, & Corbett, 1989)[8]. In the case of the stepwise condition of the NMD task, achieving greater efficiency is an expected advantage of the immediate feedback central to that condition.

[8] Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP Tutor. *Cognitive Science*, 13, 467-506

Because of the nature of the feedback in the stepwise condition, an additional performance measure representing their accuracy and efficiency within each trial can be tracked for each participant. This measure is the proportion of steps within each trial for which the participant makes the next optimal allocation. In other words, this measure is the proportion of the time the stepwise feedback did *not* have to be given to alert participants to a step that was off the optimal path. Figure 8 shows the average of this measure over all participants in the stepwise feedback conditions across all trials of training. Although the learning curve here is somewhat bumpy, there is a significant upward trend that is well fit by both a linear and power curve. Also, note that the vertical line indicates the switch between block 1 and block 2, which not surprisingly matches one of the decreases in the curve.

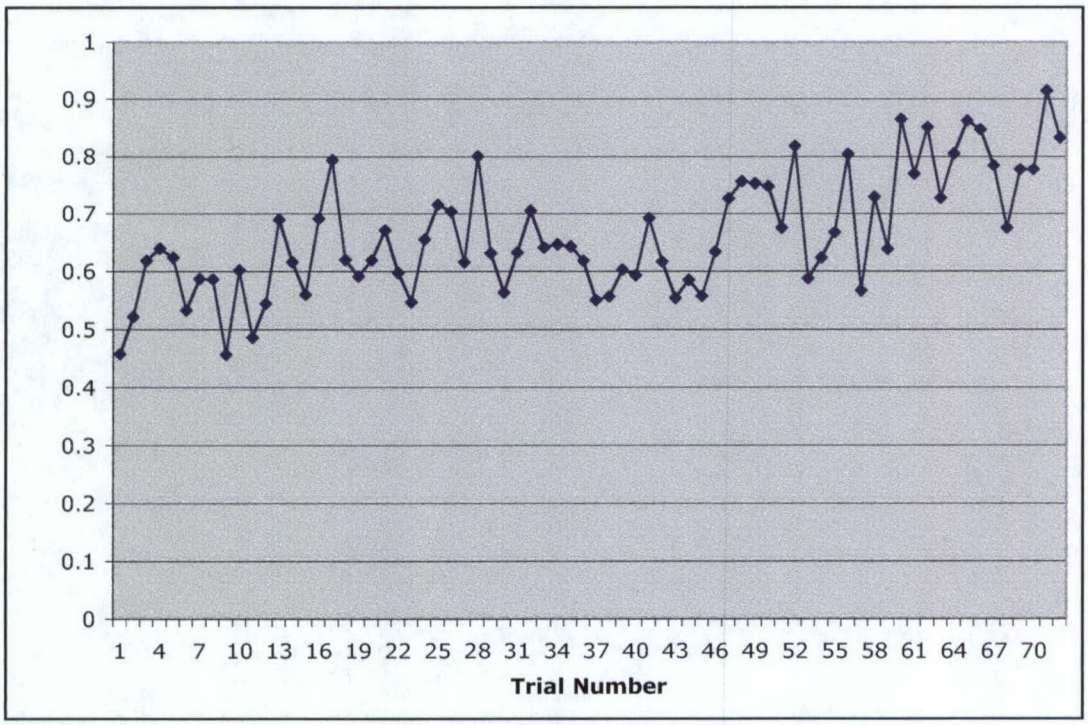

Figure 8. Learning curve across trials for stepwise feedback condition.

This measure can be conceived of both as an accuracy measure and an efficiency measure. It is an accuracy measure because it calculates the proportion of steps a participant has followed the optimal algorithm of always allocating the next GBI that will produce the largest increase in lives saved. This measure also assesses efficiency because the higher the proportion of steps that are on the optimal path the more systematically and efficiently the participant is following this strategy for optimal solution.

This measure of optimality on a step-by-step basis should be correlated with our end-of-trial measure of distance from optimal solution, so as a check on our variables and as a means of establishing the validity of this step-by-step accuracy measure, Figure 9 plots these two variables for each of the participants in the stepwise feedback condition. A linear trend line has been added, $r = -.76$. Note that this correlation would be predicted to be negative rather than positive

because the step-by-step measure calculates accuracy – higher is better – whereas the end-of-trial measure calculates distance from optimal – lower is better.

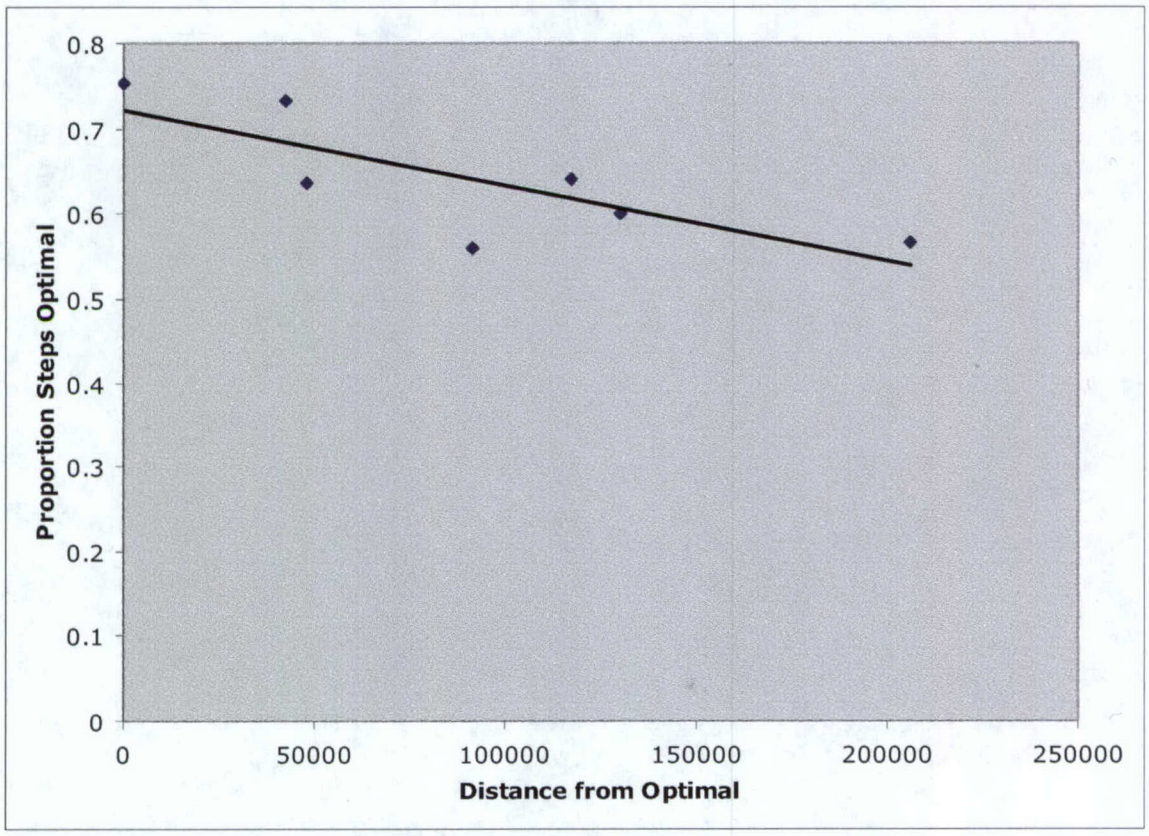

Figure 9. Association between two measures of accuracy for the stepwise condition.

Individual Differences

Along with the NMD task, participants completed several additional tasks designed to measure various individual differences in cognitive processing and other differences in aptitude and past experience. These tasks included short questionnaires asking about participants' Math SAT score and previous coursework in math and statistics/probability, a working memory span task (Engle, Kane, & Tuholski, 1999)[9], and a risk-taking assessment questionnaire.

The question for this aspect of the research was whether any of these individual difference factors would predict to some degree participants' ability to produce optimal solutions. The results in this regard did not provide strong evidence for any some relationships. All of the correlations were below .32. The strongest, at r=-.319 was between number of math courses and distance to optimal, where a negative correlation would make sense. Still, this is a rather weak relationship, as can be seen in Figure 10.

[9] Engle, R.W., Kane, M.J. & Tuholski, S.W. (1999). Individual differences in working memory capacity and what they tell us about controlled attention, general fluid intelligence and functions of the prefrontal cortex. In Miyake, A. & Shah, P. (Eds.) *Models of working memory: Mechanisms of active maintenance and executive control.* London:Cambridge Press.

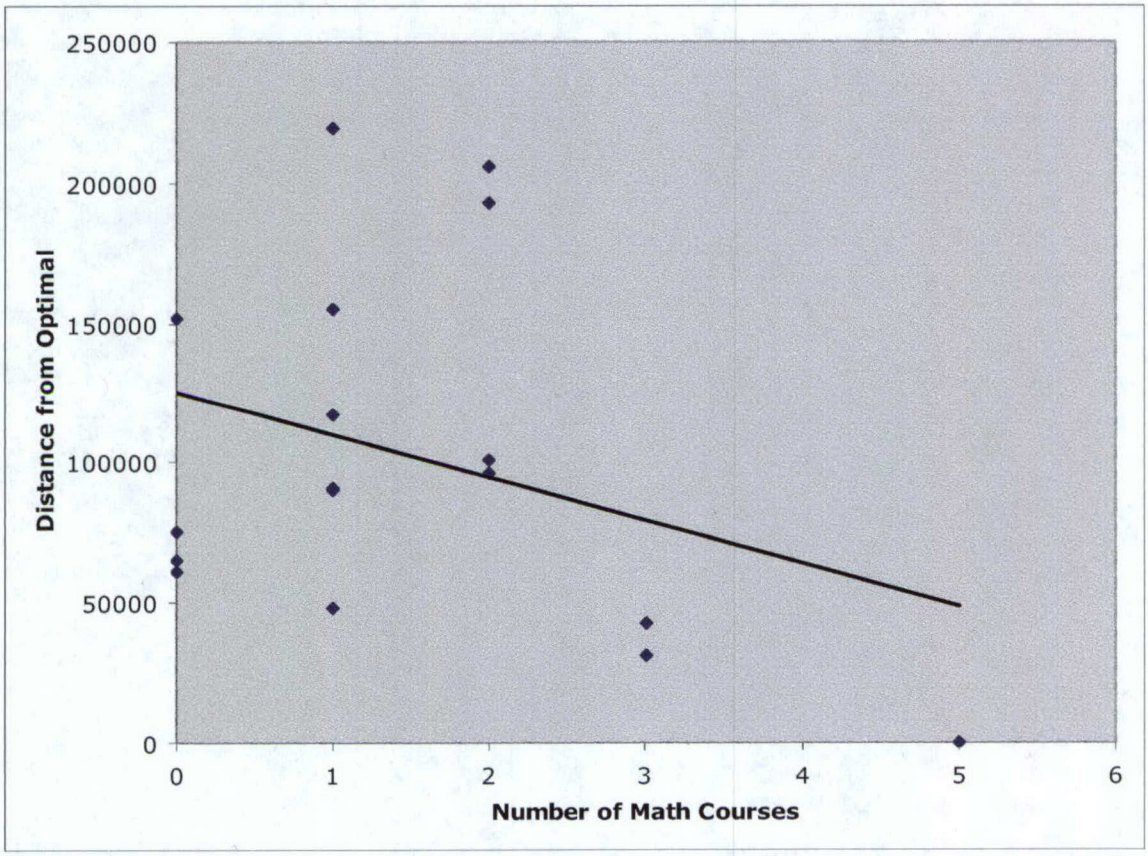

Figure 10. Association between math courses and distance from optimal.

Modeling the NMD Task

One of our primary goals in modeling the NMD task is to investigate the different consequences of a normative vs. an expert model. These two approaches may lead to different performance of the model itself, and as the basis for a training system. An expert model may or may not perform the task as well as an optimal system, and an optimal system may or may not direct a training session as well as an expert model. This makes it important to determine the ways in which the two approaches differ and whether these differences offer different advantages or disadvantages for tutoring.

In addition, we also chose to investigate the differences between an expert model induced from empirical data and an expert model constructed using knowledge engineering. It is important to know if there is a continuum from normative to expert to novice performance or whether there are qualitative differences between the categories. The effect on training of optimal vs. expert model can be studied by switching the models while keeping the rest of the training apparatus constant, providing empirical evidence as to whether faster and more accurate performance is preferable in a tutoring model to more approximate human-like processing.

Models Developed for the Project

Pure Optimal Model

The first computational model developed as part of this project is a pure optimal computational model. The algorithm is based on a simple hill-climbing principle that can be applied to monotonic spaces like the NMD problem. Monotonic here means that the individual moves do not interact, and will always have the same outcome. For example, a missile allocated in the NMD task and then de-allocated would leave the trial in exactly the same state it was in prior to the moves. Similarly, adding a missile to a city will always increase the protection afforded to that city by the same amount regardless of what has been done elsewhere within the game.

Given this problem space, the construction of the initial algorithm was quite simple. The algorithm is a simple iterative algorithm that always seeks to change missile allocations in the direction of the greatest available gain. Since the excess missiles are initially in reserve, this means that each allocation must save more lives than the decrease in reserves costs, and that no allocation could save more than the chosen move. The algorithm terminates on the first iteration that no new assignments are made. The entire algorithm is given below:

1. Set the Best_eval to the number of lives saved by adding a missile to protect City 1.
2. Set the Best_City to City 1.
3. Repeat the following steps until no further assignment is made
   a. For each City *n*, do the following:
      i. Calculate the lives that would be saved by adding a missile to protect the city.
      ii. If the gain from adding this missile to City *n* is greater than the current Best_eval then do the following:
         1. Set the Best_City to City n.
         2. Set the Best_eval to eval n.
   b. If the current Best_eval will save more lives than it will lose by removing a missile from the reserves then assign it to the Best_City.

Rule-based Cognitive Model

Although the pure optimal model was conceptually straightforward, it was not obvious how to leverage the algorithm in a teaching situation. To investigate the utility of the algorithm for learning and teaching, we implemented the algorithm within the ACT-R cognitive modeling framework. The intention of this effort was to cast the optimal algorithm into cognitive operations, thereby providing both a task analysis and a potential mediating algorithm simultaneously. Those cognitive operations can be largely characterized within the ACT-R framework as the operation of production rules. An example of a production rule involved in the task is presented here in pseudocode:

```
If adding a missile to City B will save more lives than
adding a missile to City A then City B is currently the
best candidate location for a missile.
```

In this capacity, the ACT-R language served primarily as a programming language, but one that introduced cognitive constraints. Although this use of the architecture is somewhat unusual, it proved extremely useful from a task analysis perspective. In particular, the ACT-R architecture provides high fidelity estimates for the speed of cognitive operations. The optimal algorithm, as expressed in ACT-R, took more than a minute to complete an NMD trial using any reasonable estimates of the speed of the constituent operations. However, the original NMD task was temporally structured in a way that prevented students from interacting with the task for all but the last few seconds of a trial. The original authors of the task may have intended for students to work out much of the answer in their heads and then produce it using the interface at the end of the trial. However, the mental arithmetic involved would constitute a substantial load if forced into a short time frame. After some practice in attempting to apply the optimal algorithm to solve problems, it became apparent that the task as structured could not easily or practically be solved using the optimal algorithm, and it also raised questions about what method previous participants had been using and what precisely the form of their learning was.

The entire optimal algorithm is presented in an appendix for completeness. The performance of the algorithm, because it is not stochastic, and because time was not a serious consideration, is completely unremarkable in every way: it produces the optimal answer exactly each and every time (and thus produces completely trivial graphs and figures as well).

Instance-based Near Optimal Model

One of the primary technical objectives of this project is to investigate the utility of instance-based modeling techniques based on a cognitive architecture to quickly develop high-quality synthetic entities for training purposes. To this end, we constructed an NMD performance model using the ACT-R cognitive architecture. This model, which is based on the rule-based ACT-R model discussed above, will now be described in detail (a complete model run is presented in the appendix).

Initially, the model starts with a particular scenario consisting, in the abstract, of five cities each having a particular population and an initial allocation of missiles and a follow-on city having a given probability of a follow-on attack and a number of reserve missiles that will be used to protect the city from any follow-on attack. These facts are loaded into the declarative memory of ACT-R upon being perceived. For example, the first city in scenario 1 is represented as follows:

```
City1   99.326
    isa PERCEPT
    city 1
    pop 1839449
    miss 0
```

The model begins by inspecting the cities in the scenario in a left-to-right fashion (based on the solution method used by a human task expert), and calculating the number of lives that could be

saved by increasing the allocation of missiles to the city. A run trace of the individual productions that produce this behavior follows:

```
Time  0.000: Next-City Selected
Time  0.050: Next-City Fired
Time  0.050: Info-City Selected
Time  0.100: Info-City Fired
Time  0.100: City1 Retrieved
Time  0.100: Store-City Selected
Time  0.150: Store-City Fired
```

These productions, "Next-City", "Info-City", and "Store City" are involved in perceiving the city and transferring that perception to declarative memory, which is noted when City1 is retrieved and stored. Once the perception is complete, the next thing to do is to evaluate the impact of adding a missile. The evaluation proceeds by first attempting to retrieve a previous example of a similar evaluation (i.e., it is much easier to remember a calculation than to actually do the math, so the cognitive model attempts to recall previously doing the calculation). In this case, since the model has never encountered this problem (or any other) prior to this, it cannot retrieve the solution to the evaluation, so instead it calculates it in a production named "Failure-Eval":

```
Time  0.150: Eval-Inc Selected
Time  0.200: Eval-Inc Fired
Time  1.200: Failure Retrieved
Time  1.200: Failure-Eval Selected
Time  1.250: Failure-Eval Fired
Time  1.250: Better-Eval Selected
Time  1.300: Better-Eval Fired
```

The evaluation proceeds by comparing the potential gain of adding a missile to this city with the best possibility so far during the trial. Since this is the first city inspected it is selected as the best so far ("Better-Eval"). The same process is used to evaluate the remaining cities and the follow-on city. Based on these calculations, the first city presents the opportunity for the greatest gain, so it is selected to receive a missile:

```
 Time  7.650: Allocate Selected
Allocating GBI to City 1
Optimal move: Allocate GBI to City 1
 Time  7.700: Allocate Fired
```

In addition, the trace notes the actual optimal move though the model is not provided with this information as feedback at this point. Thus, the model has taken approximately 8 seconds to perform an initial allocation (although modeling latency was not an intended goal of this project, the latency values do, in fact, correspond reasonably well with actual behavior as a result of simply leveraging the assumptions of a cognitive architecture). The model continues allocating missiles as the trial continues, trying to decide when to leave the rest of the missiles in reserve for a follow-on attack. In this trace, after working on the problem for 39 seconds, the model allocates one missile more than it should have:

```
 Time 38.550: Allocate Selected
Allocating GBI to City 5
Optimal move: no allocation (keep reserve level)
 Time 38.600: Allocate Fired
```

The model completes the trial, choosing not to allocate any more missiles, and then receives trial feedback indicating the chosen and desired (optimal) allocation of missiles to the cities:

```
 Time 45.150: No-More-Missiles Selected
 Time 45.200: No-More-Missiles Fired
 Time 45.200: No-Allocate Selected
Desired missiles for city 1 population 978332 missiles
allocated 2: 2
Desired missiles for city 2 population 344677 missiles
allocated 1: 1
Desired missiles for city 3 population 250900 missiles
allocated 0: 0
Desired missiles for city 4 population 1821557 missiles
allocated 2: 2
Desired missiles for city 5 population 1728296 missiles
allocated 3: 2
Desired missiles for follow-on city 6 population 2368000
probability   0.25 missiles left 0: 1
 Time 45.250: No-Allocate Fired
```

In this problem, the model has allocated 3 missiles to city 5 while not leaving any missiles in reserve to defend from a follow-on attack. The model processes the rows of the feedback one at a time, inspecting each for the chosen and desired coverage for that population. The first city is given the appropriate level of coverage and the model notes this:

```
Time 45.250: Next-Feedback Selected
Time 45.300: Next-Feedback Fired
Time 45.300: Feedback-City Selected
Time 45.350: Feedback-City Fired
Time 45.715: Feedback5 Retrieved
Time 45.715: Store-Feedback Selected
Time 45.765: Store-Feedback Fired
Time 45.765: Right-Allocation Selected
Time 45.815: Right-Allocation Fired
```

The more interesting case is that of the fifth city, where the model notes that an incorrect allocation was made:

```
Time 47.139: Next-Feedback Selected
Time 47.189: Next-Feedback Fired
Time 47.189: Feedback-City Selected
Time 47.239: Feedback-City Fired
```

```
Time 47.521: Feedback9 Retrieved
Time 47.521: Store-Feedback Selected
Time 47.571: Store-Feedback Fired
Time 47.571: Less-Allocation Selected
Time 47.621: Less-Allocation Fired
```

Correspondingly, the model also notes that more missiles should have been kept in reserve:

```
Time 47.621: Next-Feedback Selected
Time 47.671: Next-Feedback Fired
Time 47.671: Feedback-City Selected
Time 47.721: Feedback-City Fired
Time 47.983: Feedback10 Retrieved
Time 47.983: Store-Follow-On Selected
Time 48.033: Store-Follow-On Fired
Time 48.033: More-Reserve Selected
Time 48.083: More-Reserve Fired
Time 48.083: End-Feedback Selected
Time 48.133: End-Feedback Fired
```

The effect of the "Less-Allocation" and "More-Reserve" productions is to store an instance in declarative memory that corresponds to the difference between the action that was actually taken and the action that should have been taken (the optimal choice). This instance, a chunk of information now stored in declarative memory, will have the opportunity to be retrieved the next time the model attempts to evaluate the alternative allocations that can be made during a trial (as was performed by the "Eval-Inc" production above). The assumption that is latent in this process is that the learner is consciously choosing to attend to the feedback screen, and that the same process would not necessarily be engaged by a more passive or less problem focused viewing of the information. This assumption is borne out to some extent by the differences in performance observed in the three feedback conditions – the summary screen detailing the difference between actual performance and ideal performance appears to be a key driver of successful learning in this task.

Relating Model Performance to Human Performance

The preceding section describes the qualitative aspects of the performance model – the actions taken and their sequencing. This provides the first level of model performance description – it demonstrates that the model performs the task and does so in a way that does not overtly violate the constraints of human performance. (This step is, in fact, a prerequisite step in model validation that is often skipped before commencing with a quantitative analysis of second-order model effects. That is not to underrate the importance of finer-grained distinctions; it is simply to point out that the finer details of model correspondence are irrelevant if the broader details are wrong.)

Quantitative issues of interest include describing and explaining the proportion of time the model stays on the optimal path, and the parameters that impact that performance, and issues surrounding learning. We chose to investigate the use of two parameters in tuning model

performance: the accuracy of feedback (an environmental parameter, not a parameter of the model itself), and the accuracy of mathematical calculations. No other parameters were manipulated in the course of model development – the parameters of the ACT-R architecture were all set to either initial values or to a predetermined value prior to exploring the relationship between human performance and model performance.

The first parameter, accuracy of feedback, is of interest for two reasons: 1) we determined that the version of the task in existence prior to this project had a bug in it which had often resulted in incorrect feedback in a prior study, and thus produced noisy feedback, and 2) expert feedback is likely to be at least slightly sub-optimal, so slightly noisy optimal feedback can be used to approximate the impact of expert feedback on training. To investigate these parameters, we conducted a search across the parameter space of these variables with the intention of identifying a portion of that space that corresponded to a range of human performance observed. Data collection consisted of running a single model through a complete experimental session for each of the parameter values. Table 1 below shows the percentage of moves made on the optimal path while varying the feedback accuracy and noise in mental calculations:

Table 1. Proportion of optimal moves as a function of calculation noise and feedback noise.

	Calculation Noise					
Feed. Noise	0.5	1	2	2.5	3	Mean
2	0.55	0.53	0.54	0.56	0.51	0.54
1	0.57	0.53	0.55	0.57	0.52	0.55
0.5	0.63	0.65	0.62	0.61	0.56	0.61
0	0.66	0.64	0.62	0.62	0.59	0.63
Mean	0.60	0.59	0.58	0.59	0.55	

This space roughly corresponds to the range of human performance observed during the empirical studies, with more moves falling on the optimal path at lower feedback and calculation noise levels. The values are surprisingly consistent given that this is, in effect, a single subject study. The graph in figure 11 below shows the same data as a surface, showing that these parameters have roughly equal effects and combine in a linear fashion:

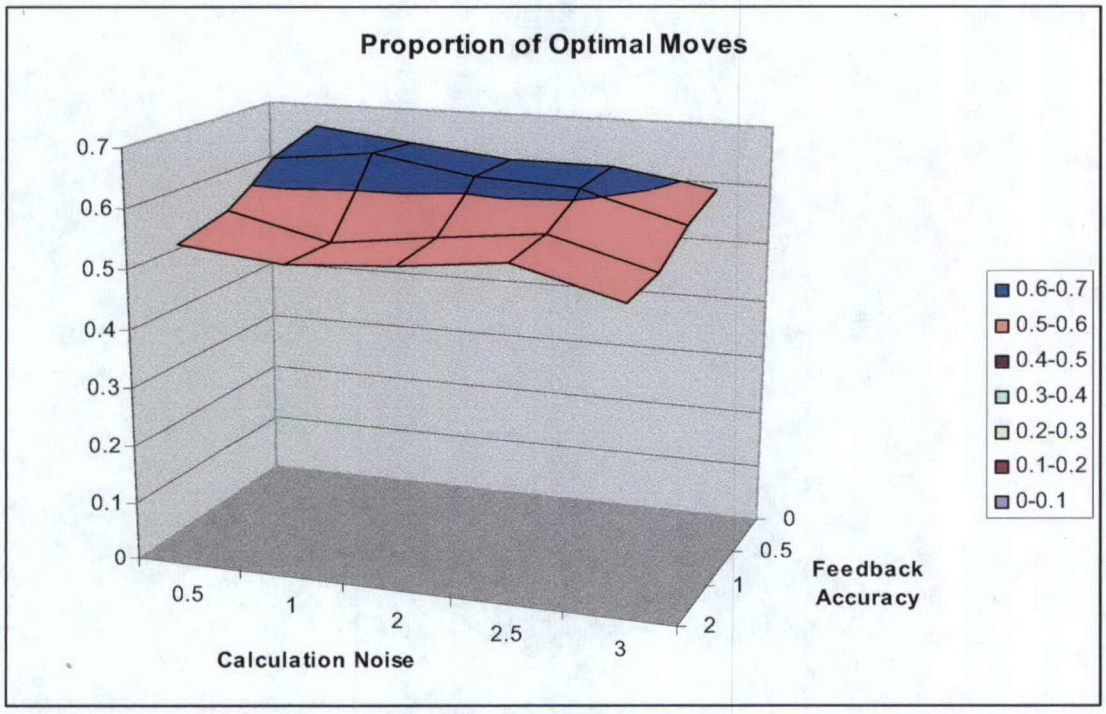

Figure 11. Parameter space relating proportion of optimal moves to calculation noise and feedback accuracy.

Learning in the NMD Model

The NMD model described above is a learning model in the sense that it does accumulate knowledge. As trials progress, the model depends more and more on recalling previous items rather than calculating the proper choice of action. This transition from calculation to memory retrieval, however, is not necessarily what many people think of when they identify learning. The model is, in fact, performing a task it can already perform – it is just doing it better over time. It might be labeled as "simply memorization", but in fact it does go beyond that.

The graph in figure 12 below shows the learning performance of an instance-based model of the NMD task, plotting trial number against percentage above optimal performance. This graph uses the same time scale and performance measure as the human performance data presented in figure 8. Like the human data, the model shows a somewhat bumpy but clearly increasing trend. The simulation was completed on the same trials as the laboratory experiment and aggregated across 20 model runs, and thus represents nearly identical conditions for both the human solvers and the cognitive models, allowing for straightforward comparison of the human performance and the cognitive model performance.

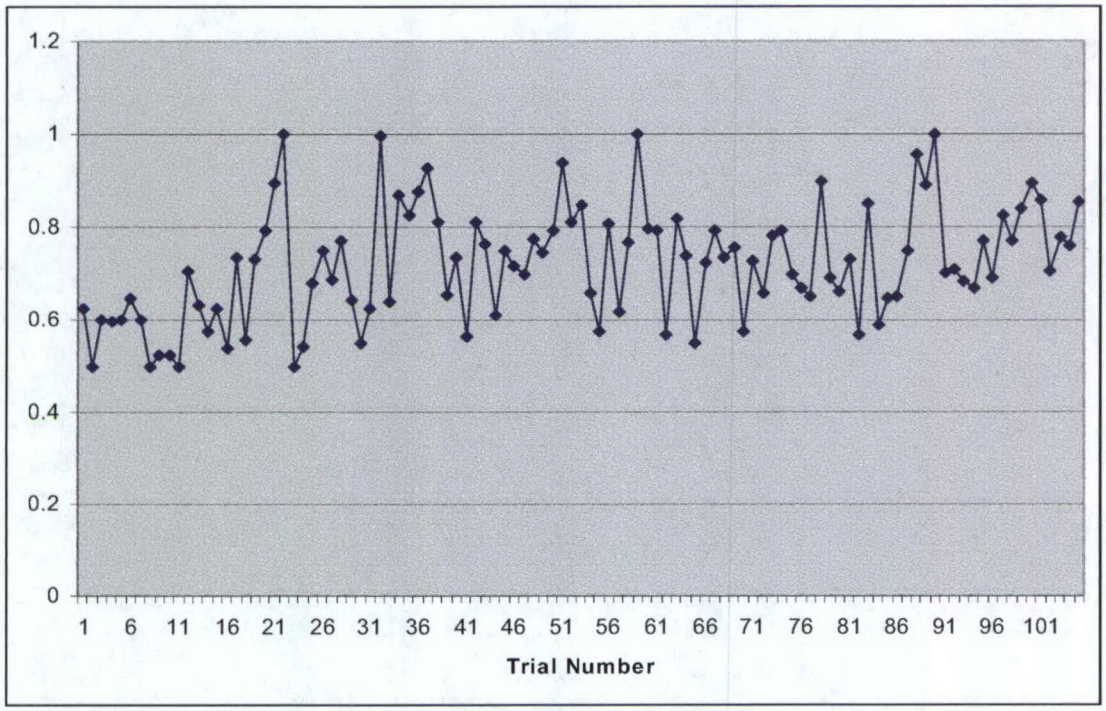

Figure 12. Learning effect produced by model in terms of proportion of optimal moves by trial.

There are two important points to be made with the graph above. The first is that learning within the ACT-R framework, represented here as the transfer of operations from a procedural, algorithmic form into a memory-based pattern matching process, captures the quantitative and qualitative aspects of the learning demonstrated by the human participants. The second point is that the learning demonstrated by the model is constrained by the representation chosen to learn within. Both of these topics will now be addressed in greater detail

The learning that takes place during an experimental session involves, qualitatively, the transfer of the seat of performance from a set of rules that algorithmically explain how to perform the calculations necessary for the task to the declarative memory of ACT-R. This has the flavor of what has been termed "Recognition Primed Decision Making", but rather than a vague notion of expertise having to do with recognition, it is made explicit here: expertise is the accumulation of task knowledge instances that allow the expert to make ever finer discriminations.

In this case, the ACT-R model is learning how many missiles go with a city of a particular size, and that begs the question of what it should be learning. In truth, the answer to that question depends on whether the goal is to model human performance, or whether there is an absolute performance goal or criterion to reach. In any case, from the perspective of the optimal algorithm the number of missiles depends completely on the relative needs of the other cities presented in the scenario. A city of 100,000 people, for example, may receive two GBIs if the remainder of the other cities in the scenario includes less than 50,000 people, but that same city might receive no GBIs if the other threatened cities all had populations exceeding a million people.

Representational Choices

The model was developed with two free parameters: feedback accuracy and mental arithmetic accuracy. We initially expected that learning effects exhibited by human task participants could be captured, if somewhat unsatisfactorily, by a systematic variation of the mental arithmetic accuracy parameter. However, the learning exhibited by the model based purely on memory effects captures both the time scale and the qualitative shape of the learning demonstrated by students in the laboratory.

The representation itself, though appearing impoverished, is actually sufficient for performing the task at a high level of competence. The following chunk is an example of the actual content of the learning:

```
Increment417    1.417
    isa INCREMENT
    pop 1738842
    miss 2
    eval 0
```

This particular fact is used by the model to decide that, if confronted with a city of population 1,738,842 that already has two missiles allocated, it is best to leave it at that (this is how the model interprets the meaning of "eval 0" – the number represents an estimate of the number of lives that would be saved by increasing the allocation, or 0 if it would be expected to cost more lives than it saves). These incremental solution improvement chunks are created during the feedback session, and though they are initially difficult to retrieve and sparse (there are many situations there is no relevant memory for early on), they eventually come to cover the problem space and provide support for good performance.

Optimal Feedback versus Expert Feedback

The simulation studies conducted examined learning with varying amounts of noise added to the conclusions reached by automated tutor. That is, though the cognitive tutor is unaware it is doing anything other than acting perfectly, it is in actuality being reduced in fidelity to compare with higher fidelity representations.

The results from the simulation studies indicated that, while small differences in feedback quality (like those expected between optimal and expert performance) have small impact, the impact on performance tends to scale with the difference in quality between optimal and actual feedback.

That said, the simulation results also support the use of actual expert performance to drive the development of a cognitive model. The initial actions taken by the current model, though based on an optimal algorithm, contribute nothing to the eventual learning. The optimal rule-based model simply provides a framework to get the model into the task, where the learning is actually memory based and is produce by *making mistakes*, not by performing correctly. That is, the mistakes are exactly what provides the opportunity for feedback. The difference between what was done and what should have been done is what forms the basis for all of the learning exhibited by the model.

Instant Feedback versus Delayed Feedback

The role of instant feedback is interesting to consider from the perspective of model building. For participants engaging in that condition, a mildly annoying ring tone would interrupt them every time they made a move that was sub-optimal. This provides good motivation for staying on task, since sloppy assignments tend to bring out the rings. However, the more interesting question is, just what are participants learning at this time in the study? Blame assignment is something that is typically considered within the very tight timeframes typically used in research within the learning community.

General Discussion and Conclusions

The modeling effort suggests that the actual learning may have been improved memory discriminations rather than any type of procedural or algorithmic improvement (and certainly demonstrates that this kind of learning can account for the change in human performance over time). However, deriving a mechanistic account for the impact of instant feedback on latency is not nearly as obvious. The instant feedback may have served to simply keep the students on course, thereby decreasing latency, but may not have had much differential impact on quality of performance because what was being learned remained the same in both conditions. The instant feedback appeared instead to improve the focus of the learning by reminding the student to stay on course. That is, it produced more efficient training – an equivalent learning effect in significantly less time. One interpretation of this is that the instant feedback kept students exploring the part of the problem space that was relevant to improved performance, improving efficiency by culling many potentially unproductive explorations that students might have otherwise engaged in. The potential, if this interpretation is correct, is that instant feedback may improve efficiency of training and allow more training to take place per unit time. While all participants received the same training in the study reported here, the outcome might have been different if the study were structured in a way to allow students to take advantage of training efficiency.

Future Directions

Research on the high-order aspects of human information processing that contribute to skilled human performance demands a quantitative description of the information processing that specifies how people learn, recognize, assess, and make decisions about events occurring in dynamic environments.

Modeling of human behavior in these dynamic environments has been pursued as a means of studying the interactions of these various human capacities. However, much of the current modeling efforts directed at representing human behavior have focused on laboratory tasks which are typically time-constrained (e.g., discrete trials of a maximum length), and take place in an extremely controlled environment with a relatively simple structure. This makes the generalization of many of these models to larger scale domains suspect. In addition, the interactions between many of the cognitive mechanisms studied in the laboratory are largely unknown due to the divide and conquer approach typical of laboratory experimentation, where the world is carved up into ever finer distinctions.

The NMD task studied in Phase I of this research is typical of small-scale laboratory experimentation. Although the task is a dynamic problem solving task, the interactions are temporally constrained such that an individual problem solving episode spans no more than several minutes. Further, the task demands clearly specify the goal of the problem solver. In Phase II, we hope to extend this work to more dynamic, interactive environments in which the behavior of trainees is less constrained and the possible goals and means of achieving them are more numerous.

As an example domain characterized by complexity and dynamic interactions, consider the tasks involved in guiding unmanned aerial vehicles (UAVs). A typical mission execution requires dynamic planning and re-planning of routes, fine motor control, and close integration of motor, perceptual, and cognitive elements within the various subtasks a pilot must perform.

Individual tasks performed by a pilot (e.g., reconnoitering a location during a reconnaissance mission) are typically specified at the level of laboratory experiments (i.e., the complexity that would be typical of a single trial during an experiment). However, the overall tasks performed by pilots are better described as occurring at a higher level of aggregation, where the subtasks are assembled according to little-studied constraints and processes.

This recognition – that we need a way to assemble behavior described at a low level into aggregate behavior – has led us to search for a potential stand-in for the actual processes used by human solvers that could produce very similar behavior with much less effort than, for example, knowledge engineering. In our previous work (detailed below), we have characterized human performance in terms of an optimal model given human constraints. In general, the approach has demonstrated that an optimal model augmented with human constraints provides a reasonable model of skilled human behavior. This suggests that this method could be applied to the problem of sequencing cognition in general. That is, the assembly of individual tasks could be achieved through the use of a planning system typical of the Artificial Intelligence and Operations Research communities that is capable of sequencing the broader level of cognition through an optimal or near-optimal approach to the problem (note that the Apex system approaches sequencing the finer level of cognition in this way, but does not provide a robust account of broader cognition). However, simply using an unconstrained optimal planner would likely result in behavior that is not plausible. What is needed, instead, is a set of constraints on human planning capabilities that can be used to constrain the planning system.

Fortunately, many researchers have realized that the interactions of individual cognitive components are sometimes as important as the individual parts, and that those interactions will be exposed by necessity in any complex, dynamic domain. Thus, there has been a recent emphasis on complex, dynamic domains that necessitate the integration of many skills within a single cognitive model to enable performance in that problem domain. This integration, however, is laborious. To put the findings of specific experiments back together we require a methodology for assembling these component cognitive models that can be accomplished with less effort than the current state of the art. That is, what is needed is a methodology for transitioning the high-fidelity models used in laboratory experiments to the dynamic environments typical of real-world complex tasks.

A candidate approach to bridging the divide between high-fidelity cognitive modeling and performance in large-scale dynamic domains is the induction of small-scale cognitive models from optimal or near-optimal algorithms and the use of an optimal or near-optimal planner in selection of appropriate small-scale models during performance.[10] In addition to inducing a cognitive model from an optimal algorithm, alternatively, it is also possible to induce a cognitive model from an optimal performance. This proposal focuses on developing these methods and applying them to create robust models of human behavior for use in training and simulation systems. The key problems in development of a large-scale cognitive model include:

- successful induction of the goal structure so individual actions can be properly interpreted,
- segmentation of a stream of continuous action into discrete cues and actions taken, and,
- integrating logical sequencing of actions at a broad scale (i.e., planning) with the finer temporal scale of those individual actions.

As a whole, our future research plans comprise two distinct threads. The first thread will involve classroom research on individual student choice intended to determine constraints on human planning behavior that delineate its departure from optimality, integrating modeling using the methodology developed in Phase I of this research. Part of the classroom work will help explain why and where people diverge from optimal strategies. In particular, some of this classroom research will explore the connections between individual difference variables and strategy choices. These relationships and more general results on why/where people don't make optimal choices will be used to inform work on the second thread by circumscribing the relation of student behavior to optimal planning.

The second thread, which will be pursued simultaneously with the first, will involve a modeling project that applies the Phase I methodology (i.e., lessons learned about the use of optimal algorithms as the core of a training system) to existing expert performance data in a highly complex, dynamic environment – UAV operation. This thread will focus on integrating an optimal planner with a cognitive modeling environment (the environment that supports the tutoring models used in Phase I). The knowledge obtained from the classroom study will be leveraged to inform the further development of the planning system for sequencing cognitive actions and used to develop a training system in the UAV domain. The UAV models will then be cross validated using the methods from the classroom study and the data from expert performance in the task domain.

Future Technical Objectives

The problem of leveraging an optimal model of performance to sequence cognition at a finer grain size for use in both training (i.e., prescriptively) and in the development of simulation

[10] The terms optimal, near-optimal, and normative are used throughout the fields of Cognitive Science with a wide variety of definitions. For the purposes of this proposal, *optimal* means optimal with respect to the (external) task constraints, *near-optimal* means optimal with respect to the (internal) algorithm processing constraints and *normative* is used as a synonym for *optimal*. An *expert* model is a model derived from human performance, and can be expected to be largely indistinguishable from a *near-optimal* model, which can be derived from an *optimal* model. Thus *expert* and *near-optimal* models are expected to have indistinguishable performance, but different derivation methods.

models of human behavior (i.e., descriptively) can be approached by tackling the following technical objectives.

1. **Extend tutoring methodology explored in Phase I to dynamic, real-time environments**

   The methodology for tutoring developed in Phase I used a cognitively constrained optimal model-based approach to tutoring to suggest remediations. These remediations may require different modalities in more complex, dynamic environments (e.g., audible feedback for a visual task might be more effective).

2. **Apply instance-based methods to micro-cognitive model development**

   We will refine a methodology for seeding instance models with both normatively correct data (produced from an optimal model) and expert performance data based on the initial results from Phase I. Seeding instance models through the observation of ideal behavior has the potential to generalize to a large range of simulation environments, and could potentially reduce the cost of cognitive models and expand their use within these simulation environments in general.

3. **Investigate the relationship between learned optimal behavior and actual use of that behavior in real-world settings**

   Although students in real-world education-related choice situations can demonstrably learn optimal choice strategies, they do not necessarily use those strategies when they should. That is, although they learn the rules, they fail to apply them in the correct context (discussed in further detail below). The purpose of this objective is to understand the causes of this failure and develop potential remediations for the learning environment.

4. **Investigate optimal planning approaches to specifying macro-cognitive behavior**

   Although human behavior is expected to be sub-optimal, an optimal planner can potentially be used as the core of a system for simulating human behavior, especially for sequencing cognition at a broad scale (i.e., sequencing the action of micro-cognitive operations).

5. **Explore methods for unifying instance-based micro-cognitive models with rule-based, planning focused macro-cognitive models**

   The methods developed for producing micro-cognitive instance-based models and rule-based planning focused macro-cognitive models exist at different temporal scales. Phase I explored modeling and tutoring micro-cognitive actions; this objective would explore ways of linking individual models created using the Phase I methodology. Integrating these models requires a method for transferring constraints from the micro-cognitive models to the macro-cognitive models for use in planning, and a method for the macro-cognitive system to transfer control or sequence the action of micro-cognitive models for use in execution.

6. **Explore key attributes of instance recording, selection of features and attributes**

   The density at which human performance is recorded has a direct bearing on the development of models for simulation and training. We will explore the benefits of

extremely high density recording and the impact of that density on the segmentation of training instances (i.e., determining what constitutes an instance in a largely continuous data stream).

Extending instance-based modeling using a global planner

Instance-based modeling techniques, which are inherently learning techniques, depend on a collection of instances of behavior that the learning process can use to tune its behavior to a closer and closer approximation of the source behavior. The source data for instance-based modeling techniques can be either expert-level performance, or optimal model performance, or self-performance with feedback relating the effectiveness of the actions taken. An emerging paradigm within the ACT-R community appears to be the development of instance-based micro-models of decisions that are then unified with a rule-based approach to higher-level cognition.[11] This approach suggests the development of multi-level models of individual behavior where the global level is capable of roughly directing the behavior of the individual, while the local level is based on perceptual and motor constraints as they interact with learning and memory (see figure 13 below).

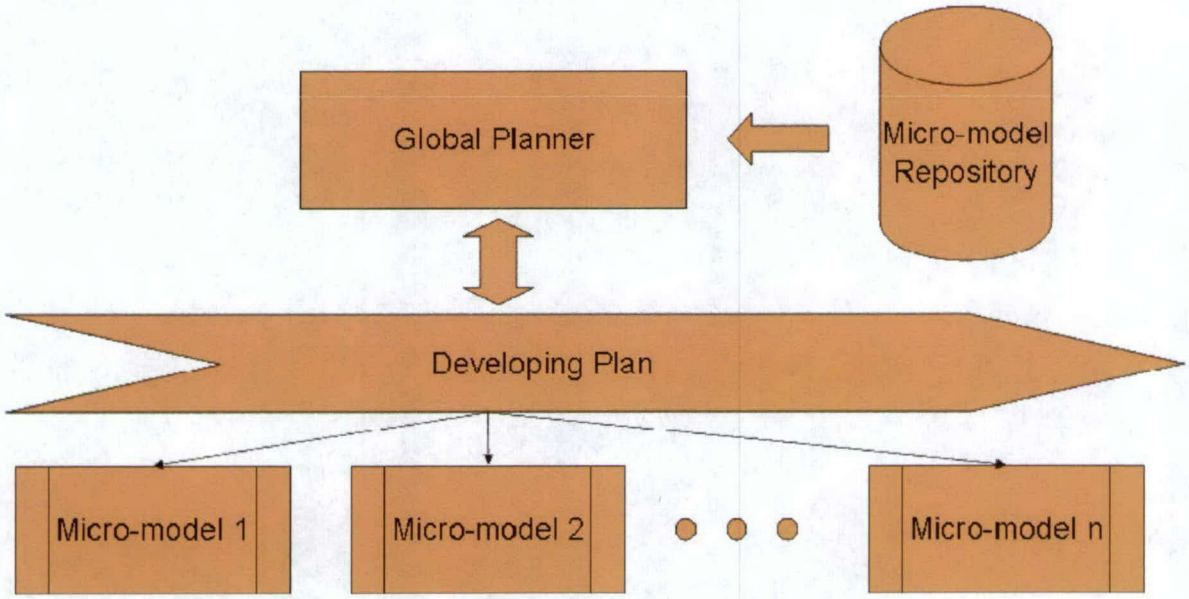

Figure 13. A Global Planner the Sequences Micro-models.

The global level of behavior is, conveniently for the modelers, often given as a task constraint. That is, in typical experiments, the task is focused on a trial-by-trial format where the experimenter directs the participant in the pursuit of a particular goal, so the goal and task structure is known *a-priori*. This bottom-up approach to high-level cognition has resulted in a paradigm where cognitive modeling environments and approaches often generate PERT charts of their activity (e.g., GOMS, APEX, ACT-R/PM), but the sequencing of the activity is done

[11] Gonzalez, C., Lerch, F. J., & Lebiere, C. (submitted). Instance-based learning in real-time dynamic decision making. Submitted to *Cognitive Science.*

without planning across possible outcomes. That is, all of the planning is incorporated into the local decisions, rather than having a more global plan that constrains the sequencing of the micro operations (at a deeper level, of course, the architectures themselves provide sequencing for individual operations, but that is at a finer grain scale than being discussed here). The result is that many cognitive models have very little representation of high-level planning that might be used to sequence an overall solution to any abstract problem. A secondary outcome of this has been an avoidance of domains characterized by long sequences of operations and sustained operations. This also leads to the curious issue that, though these cognitive architectures use the descriptive language of the planning domain to describe their actions, there is little attempt to actually plan those actions using a plan representation as is done by planning systems (e.g., in hierarchical planning, partial order planning (POP), the GraphPlan planning system, etc.), which were originally developed based on the behavior of humans organizing complex behavior.

Cognitive models often fail to represent planning at a high level, and this failure is typical not just of the models that simulate human behavior, but is also typically a missing aspect of the architecture that supports the model. As a result, cognitive models are often not applied to complex tasks such as sustained operations, since there is little theoretical basis the architectures can bring to the extensive planning typical in these domains. What is needed to address this weakness is the incorporation of more abstract models of problem solving that can be used to guide global behavior. Although this is commonly done in areas such as computer science, logistics and operations research (e.g., through planning and scheduling algorithms), the cognitive science community has not widely adopted planning algorithms as a method of sequencing high-fidelity, local behavior (exceptions include work such as Koedinger and Anderson, 1990[12], where they demonstrated that a diagrammatic representation of problems directly supported forward-chaining solution approaches).

An open research question is how to determine the global level of a system for a cognitive model in a less well-structured task where the task structure is not known or given. One candidate approach is the use of a planning system that is subject to simple cognitive constraints and is designed to work with abstract graph search representations to guide the global behavior. This requires knowledge, however, of the relationship between human behavior and optimal choices. That is, we must know something of those constraints before we can use them. Thus, we will now detail some of the necessary work to expose those constraints.

Exposing human problem solving constraints

We expect this research to lead to a greater understanding of the relationship between optimal algorithms and human behavior, especially in regards to strategy choice and planning. As a prerequisite to detailing our approach to pursuing this research, we will first discuss some of the relevant research already conducted on choice behavior and leverage that to motivate several of the technical and research objectives that follow.

[12] Koedinger, K. R., & Anderson, J. R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science*, 14, 511-550.

Since Newell and Simon's seminal work[13], problem solving has been studied as a process of making choices. One theme in their work and subsequent work is that problem solvers tend to make effective choices. For example, means-ends analysis and hill climbing are two problem-solving heuristics that people naturally use that work well in many routine, everyday problem situations.[14] In addition, when solvers have repeated opportunities to work problems in a given domain, they learn to choose the strategies and problem-solving operators that lead to greatest success.[15,16] This suggests an adaptive choice process is at work. Indeed, the ACT-R cognitive architecture posits a choice mechanism that usually selects the utility-maximizing problem-solving operator, where operators' utilities have been learned through experience. More specifically, at each step in problem solving, the ACT-R mechanism chooses according to a soft-max rule[17] by virtue of a noise parameter added into the process. The probability of choosing production i is given by:

$$P(i) = e^{U_i/\tau} \Big/ \sum_j e^{Uj/\tau}$$

Where U_i is the estimated utility of the production i and τ is a noise parameter.

Lovett (1998) showed that in non-stationary environments (i.e., situations when operators might change their probability of success) this noisy choice process leads to the best overall performance of the model compared to several competitors including a perfect, "noise-free" choice process.

In contrast to the adaptive behavior of participants in these laboratory experiments – generally college students participating for pay or course credit – there is evidence that similar students, when making choices among problem-solving and learning strategies in real classrooms, do not make effective choices. Research on metacognitive strategies, for instance, indicates that students do not apply effective study skills even after training in those skills.[18] That is, when trained in an optimal strategy, and after demonstrating competence in that strategy, students do not actually engage in that strategy. Another line of research on how students use worked examples as study aids shows that only a portion of the students think through why each step was taken in the worked example, a strategy that greatly benefits them in subsequent problem solving.[19] The remaining students use other, presumably less effective strategies for studying the worked examples and then perform more poorly on subsequent problems.

---

[13] Newell & Simon. (1972). *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall.

[14] Even though hill climbing can lead to impasses in problems with local minima, it is an extremely low-cost heuristic that is still adequate to a wide variety of tasks.

[15] Lovett, M. C., & Anderson, J. R. (1996). History of success and current context in problem solving: Combined influences on operator selection. *Cognitive Psychology*, 31, 168-217.

[16] Lovett, M. C. (1998). Choice. In J. R. Anderson & C. Lebiere (Eds.) *Atomic Components of Thought*. Mahwah, NJ: Erlbaum.

[17] Luce, R. D. (1959). *Individual Choice Behavior: A Theoretical Analysis*. New York: Wiley.

[18] Hattie, J., Biggs, J., & Purdie, N. (1996). Effects of learning skills interventions on student learning: A meta-analysis. *Review of Educational Research*, 66, 99-136.

[19] Chi, M. T. H., Bassok, M., Lewis, M. Reimann, P., & Glaser, R. (1989) Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13, 145-182.

The natural question then is, why are students in real-world, education-related choice situations not showing the same tendency to learn and use more effective strategies as they show they are able to do in the laboratory? Why, when trained in a highly effective strategy, do they choose a less effective strategy? Understanding the answer to these questions is essential to any field application of techniques based on optimal choice algorithms.

This project has suggested exploring these questions by a combination of approaches:

1) Gathering a rich data stream from students taking a real course for a grade so that the strategies they are choosing and the effectiveness of those strategies can be inferred from the data. Strategy determination is often difficult with the sparser data collection – rich data collection is a prerequisite for identifying the strategies used by students.
2) Building a set of computational models within the ACT-R system that include a variety of possible strategies so that the effectiveness of these strategies can be evaluated in a simulation context under the natural constraints of the human learning system.
3) Collecting a set of individual difference variables (e.g., working memory capacity, prior knowledge, epistemological believes, speed-accuracy trade-offs) from the students taking the course to explore the role of individual differences in students' strategy choices and the effectiveness of different strategies.
4) Exploring the role of these individual difference variables in the computational models to identify how the effectiveness of different strategies varies as a function of particular individual difference variables.

Note that the preceding four steps act to describe and explain the degree to which students' choices are effective and to identify ways in which their choices are not effective. This leads naturally to a fifth step:

5) Developing and testing new interventions to diagnose and remediate situations where students are not making the most effective learning choices.

Integrating human problem solving constraints into a planning system

Given that we have determined key constraints in planning, the next step is to integrate these constraints with a planning system. Previously, we developed a planning system that incorporated aspects of Means-Ends Analysis and which borrowed from the CHREST architecture of Gobet (1997)[20], which was created to explain the search behavior of chess players. We have already demonstrated (Best, 2004)[21] that it is possible to use the same abstract graph planner used for solving the TSP to guide the global level of behavior for autonomous robotic navigation within an interior space. In that case, however, lower level decision processes (i.e., the perceptual and motor decisions) were not instance-based, but were instead rule-based, which exposed issues of brittleness and intractability (from a knowledge-engineering

[20] Gobet, F. (1997). A pattern-recognition theory of search in expert problem solving. *Thinking and Reasoning*, 3, 291-313.

[21] Best, B. J. (2004). Route Planning and Threat Avoidance through Cognitive Robotics. Presented at *Winter Simulation Conference.*

perspective). This, in turn suggests that the lower level decision processes might be more properly modeled using an instance-based approach.

Following this approach, it should be possible to build a complete cognitive model of a complex task simultaneously approaching it from the bottom-up and the top-down. To make this intuition concrete, consider the task of controlling the Predator UAV. The Predator UAV Simulated Task Environment, available from the PALM lab at Mesa AFRL[22], provides a ready environment for studying the interactions of a cognitive model with a complex task environment. In addition, the PALM lab has collected data on expert human performance across a variety of basic maneuvers involving the Predator (e.g., maintain altitude and speed, turn to a heading and decrease altitude by a specified amount within 60 seconds, etc.), and referenced these against the optimal performance in each of those maneuvers, measuring the root-mean-square differences between the human performance and the optimal performance.

The data collected at the PALM lab are worth exploring in further detail. Data collection involved sampling control settings every 200 milliseconds, which allows for a very fine level of analysis. Given this level of analysis, it is possible to build instance-based micro-models of control that focus on two issues: 1) determining the amount of motor input necessary to produce a desired control setting (e.g., how much throttle motion is necessary to set achieve 3000 RPM?), and 2) what control setting is necessary to achieve a desired change in the flight envelope (e.g., what airspeed will produce a descent rate of 2 feet per second?). The flight envelope, though primarily determined by the control settings, is also significantly influenced by environmental factors (e.g., wind) that add a component of uncertainty to the effect of any control settings. Thus, a complete micro-model of behavior for a simple maneuver must incorporate visual perceptual information (e.g., current display readout values) with motor outputs, and a feedback loop that allows interaction between the two to account for uncertainty and variability in the effects of the control settings. In addition, given the modeling paradigm discussed here, it will be straightforward to model individual differences from a signal detection perspective, and to investigate if individual performances across tasks can be captured using this methodology.

Given the data for the expert performance, we know it is possible (though not necessarily easy) to construct an instance-based model of performance for each of the maneuvers. This has in fact been done for many other tasks, some of which are quite complex – for example, by using expert play in backgammon to guide the construction of an expert-level backgammon player through a simple observation and learning paradigm. In addition, the basic maneuvers provide mid-level tasks that could be leveraged as building blocks to guide performance in a more complex task, since each of these basic maneuvers is well-specified and time-constrained. That is, the more complex maneuvers are likely composed of sequences of the basic maneuvers.

The main difficulty in inducing model structure at the global level in many domains is in decomposing the task performance – knowing what goal the participant is currently pursuing – so the overall goal structure can be induced. While protocol analysis, cognitive task analysis, and other tools exist for attempting to answer this question, it is also possible that the sequencing of instance-based micro-models of individual maneuvers by a planning system subject to cognitive constraints may provide a compelling account of decision processes in complex

[22] http://www.mesa.afmc.af.mil/html/palmlab.htm

environments, and sidestep this difficult issue and many of the inherent limitations of knowledge engineering. Given that the same planning system paradigm has successfully modeled chess playing, human approaches to the TSP, and navigation in interior spaces, we think it is possible that this paradigm could provide a framework for task performance in the UAV domain.

The general questions we are interested in pursuing are: 1) whether it is possible to use the optimal models of individual maneuver performance that the expert performance is compared with to guide the development of instance based micro-models of control for each of these tasks, and 2) whether unification of these micro-models through a cognitively constrained planning framework that orders the application of individual micro-models in the pursuit of an overall goal provides a reasonable model of overall behavior. Specifically, we are interested in applying this modeling technique to data collected for the Predator UAV simulated task environment, an environment characterized by uncertainty and high workloads across sustained operations, and which allows for the exploration of perceptual and motor constraints on decision processes in the allocation of resources. To this end, the PALM lab and Micro Analysis & Design are pursuing a CRDA to allow for the sharing of code, models, the Predator STE, and collected data.

Commercialization

The training community needs automated tutors that can be constructed within a reasonable time and cost, and existing rule-based approaches to codify the knowledge of experts fall short of achieving that goal.

The completed Phase I effort has produced an approach to cognitive modeling that extends existing systems while avoiding many of the shortcomings of the knowledge engineering approach to authoring those systems. Our approach will yield more realistic and more accurate models of normative and expert performance while minimizing the investment that must be made to produce and maintain those models.

While our primary focus in Phase II will be to develop a modeling framework and automated tutor useful to the Air Force, the resulting software and methodology will have applications in many domains. It is the ideal core for any automated training system intended to teach humans to perform complex tasks and assess their performance in sustained operations. Examples include flying an unmanned aerial vehicle, operating an emerging weapon system, and allocating ground based interceptors in a missile defense environment.

In addition to providing the basis for a tutoring methodology, this work also has the potential for providing a methodology for populating constructive simulations with intelligent agents both as teammates and opponents. Although it is difficult to estimate, the need for non-player characters in various military simulations is certainly enormous, and the potential contribution of this methodology for simultaneously providing tutoring agents, teammates, and opponents cannot be understated.

Beyond the training domain, this approach could be used in any application requiring accurate, affordable human behavior representation. For example, it could be used to implement autonomous robot navigation in an interior space. It could form the core of an intelligent assistant in a decision support system designed to aid humans operating increasingly complex

systems in a reduced manning environment. It could be used in human-computer interface design. In fact, MA&D is already working on a tool that allows system developers to evaluate and improve the design of a user interface by using an executing human behavior model to exercise it.

Our methodology has developed the concept of augmenting a normative model with human constraints to provide a representation of typical, rather than optimal, human performance on a complex task. Using a normative model augmented with human constraints guarantees that the resulting tutor is teaching something the trainee is capable of learning. Such a model could also be used to represent a believable human entity within any domain a tutor was built for, since the optimal performance model is at the core of the tutoring system. For example, a UAV pilot tutor for a Computer Generated Force environment would, by necessity, already be capable of flying the missions it was developed to train the trainee for. As a result, this approach has the potential for populating constructive simulations with believable entities at a low cost compared to knowledge engineering approaches, but at much higher fidelity than other low cost approaches (such as finite state machine behavior models).

During Phase II, we will seek input from a variety of simulation based training and human behavior representation communities in order to make design and implementation decisions that will enable the product to address a wide range of applications.

The private sector appeal is potentially even larger, but, accordingly, much harder to estimate. The most obvious products to come from this effort would be more effective automated training systems, constructed with much less time and money than is currently typical. The ability to accurately model expert and normative performance could transform something like the Microsoft Paper Clip from an annoying distraction into a truly useful assistant. This research could also have an impact on the development of computer generated avatars and non-player characters for the computer gaming industry.

The results of this research also hold the promise of addressing problems, such as has already been demonstrated for the Traveling Salesperson Problem, which are intractable using traditional computational methods, but which become quite tractable when the problem representation exploits the kinds of aggregate behaviors used by humans in solving the problem.

Much of MA&D's revenue comes from consulting services, rather than commercial product sales. We believe that we can generate significant revenue through consulting services related to the proposed work, using our expertise to develop expert and normative models and package them in the rule-based framework to create standalone training applications and human behavior representations. As an example, based on the number and variety of companies that have contacted us requesting consulting services to create believable CGF entities, we estimate that we could increase our consulting business by 10% in the first year by aggressively marketing this technology and our capabilities in applying it.

MA&D has successfully marketed and sold quality simulation products to commercial customers for over 20 years. A key part of our success is our commitment to continual product improvements, most of which are driven by customer demand, and to providing customer

support that is exemplary in the marketplace. MA&D also has an excellent track record generating significant revenue from customization, application, and follow-on projects resulting from SBIR efforts. Based on this experience, we believe that we have a realistic grasp on the potential market and our ability to find or generate the funding necessary to productize, promote, and sell the technology proposed here.

Appendices

Appendix 1: Optimal Model

```
(defvar GBI-count 10)
(defparameter *GBI-hit-rate* 0.30)
(defparameter *follow-on-probability* 0.25)

(defstruct target-city
  (name)
  (population 0)
  (GBIs 0)
  (pct-saved 0)
  (pop-saved 0)
  )

(defun create-scenario (pop-a pop-b pop-c pop-d pop-e pop-follow-on)
  (setq city-1 (make-target-city :name 'city-a :population pop-a))
  (setq city-2 (make-target-city :name 'city-b :population pop-b))
  (setq city-3 (make-target-city :name 'city-c :population pop-c))
  (setq city-4 (make-target-city :name 'city-d :population pop-d))
  (setq city-5 (make-target-city :name 'city-e :population pop-e))
  (setq city-6 (make-target-city :name 'follow-on-city :population pop-
follow-on)))

(defun calc-pct-saved (num-GBIs)
  (let ((factor (expt 0.3 num-GBIs)))
    (- 1 factor)))

(defun calc-pop-saved (population num-GBIs)
  (let ((pct-saved (calc-pct-saved num-GBIs)))
    (* population pct-saved)))

(defun calc-follow-on-pop-saved (population num-GBIs)
  (let ((attack-pct-saved (* *follow-on-probability* (calc-pct-saved num-
GBIs)))
        (no-attack-pct-saved (- 1 *follow-on-probability*)))
    (+ (* population attack-pct-saved)
       (* population no-attack-pct-saved))))

(defun calc-gain (population num-GBIs)
  (let* ((current-num-saved (calc-pop-saved population num-GBIs))
         (projected-num-saved (calc-pop-saved population (+ num-GBIs 1)))
         (follow-on-current-num-saved (calc-follow-on-pop-saved (target-city-
population city-6) (target-city-GBIs city-6)))
         (follow-on-projected-num-saved (calc-follow-on-pop-saved (target-
city-population city-6) (- (target-city-GBIs city-6) 1))))
    (+ (- projected-num-saved current-num-saved) (- follow-on-projected-num-
saved follow-on-current-num-saved))))

(defun GBI-decision (city gain-1 gain-2 gain-3 gain-4 gain-5)
  (setf (target-city-GBIs city) (+ (target-city-GBIs city) 1))
  (setf (target-city-GBIs city-6) (- (target-city-GBIs city-6) 1))
  (format t "~%Allocating GBI to ~s with following gains: ~s ~s ~s ~s ~s"
(target-city-name city) gain-1 gain-2 gain-3 gain-4 gain-5)
```

```
  )

(defun no-GBI-decision (gain-1 gain-2 gain-3 gain-4 gain-5)
  (format t "~%Keeping current allocation of GBIs with following gains: ~s ~s
~s ~s ~s" gain-1 gain-2 gain-3 gain-4 gain-5)
  )

(defun allocate-GBI ()
  (let ((gain-1 (calc-gain (target-city-population city-1) (target-city-GBIs
city-1)))
        (gain-2 (calc-gain (target-city-population city-2) (target-city-GBIs
city-2)))
        (gain-3 (calc-gain (target-city-population city-3) (target-city-GBIs
city-3)))
        (gain-4 (calc-gain (target-city-population city-4) (target-city-GBIs
city-4)))
        (gain-5 (calc-gain (target-city-population city-5) (target-city-GBIs
city-5))))
    (cond ((>= gain-1 (max gain-1 gain-2 gain-3 gain-4 gain-5 0))
           (GBI-decision city-1 gain-1 gain-2 gain-3 gain-4 gain-5))
          ((>= gain-2 (max gain-1 gain-2 gain-3 gain-4 gain-5 0))
           (GBI-decision city-2 gain-1 gain-2 gain-3 gain-4 gain-5))
          ((>= gain-3 (max gain-1 gain-2 gain-3 gain-4 gain-5 0))
           (GBI-decision city-3 gain-1 gain-2 gain-3 gain-4 gain-5))
          ((>= gain-4 (max gain-1 gain-2 gain-3 gain-4 gain-5 0))
           (GBI-decision city-4 gain-1 gain-2 gain-3 gain-4 gain-5))
          ((>= gain-5 (max gain-1 gain-2 gain-3 gain-4 gain-5 0))
           (GBI-decision city-5 gain-1 gain-2 gain-3 gain-4 gain-5))
          (t (no-GBI-decision gain-1 gain-2 gain-3 gain-4 gain-5)))))

(defun display-city (target)
  (format t "~%Name ~s Population ~s GBIs ~s Expected Saved ~s"
    (target-city-name target)
    (target-city-population target)
    (target-city-GBIs target)
    (calc-pop-saved (target-city-population target) (target-city-GBIs
target))))

(defun find-optimal ()
  (setf (target-city-GBIs city-6) GBI-count)
  (dotimes (i GBI-count)
    (allocate-GBI)
    (format t "~%Result of allocation:")
    (display-city city-1)
    (display-city city-2)
    (display-city city-3)
    (display-city city-4)
    (display-city city-5)
    (display-city city-6)
    )
  (format t "~%Total saved: ~s" (+ (calc-pop-saved (target-city-population
city-1) (target-city-GBIs city-1))
                                   (calc-pop-saved (target-city-population
city-2) (target-city-GBIs city-2))
                                   (calc-pop-saved (target-city-population
city-3) (target-city-GBIs city-3))
```

```
                                    (calc-pop-saved (target-city-population
city-4) (target-city-GBIs city-4))
                                    (calc-pop-saved (target-city-population
city-5) (target-city-GBIs city-5))
                                    (calc-follow-on-pop-saved (target-city-
population city-6) (target-city-GBIs city-6))))
  )

(create-scenario 1839449 979044 1739986 978512 267409 1260533)

(find-optimal)
```

Appendix 2: Expert (Cognitive) Model

```
;; expert NMD model

;; note - uses 'metropolis' instead of 'city' so it can co-exist with optimal
model
;;        in the same name space

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; from optimal model...
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defvar GBI-count 10)
(defparameter *GBI-hit-rate* 0.30)
(defparameter *follow-on-probability* 0.25)

(defun calc-pct-saved (num-GBIs)
  (let ((factor (expt 0.3 num-GBIs)))
    (- 1 factor)))

(defun calc-pop-saved (population num-GBIs)
  (let ((pct-saved (calc-pct-saved num-GBIs)))
    (* population pct-saved)))

(defun calc-follow-on-pop-saved (population num-GBIs)
  (let ((attack-pct-saved (* *follow-on-probability* (calc-pct-saved num-
GBIs)))
        (no-attack-pct-saved (- 1 *follow-on-probability*)))
    (+ (* population attack-pct-saved)
       (* population no-attack-pct-saved))))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun calc-initial-saved ()
  (let* ((follow-on-pop (get-slot-value (get-safe-wme 'scenario-1) 'follow-
on-metropolis))
         (reserve-GBIs (get-slot-value (get-safe-wme 'allocations-1) 'follow-
on-metropolis))
         (follow-on-current-num-saved (calc-follow-on-pop-saved follow-on-pop
reserve-GBIs)))
    follow-on-current-num-saved
    )
  )


(defun calc-expected-saved (metropolis-name)
  (let* ((metropolis (get-safe-wme metropolis-name))
         (metropolis-pop (get-slot-value (get-safe-wme 'scenario-1) (get-wme-
name metropolis)))
         (follow-on-pop (get-slot-value (get-safe-wme 'scenario-1) 'follow-
on-metropolis))
         (current-GBIs (get-slot-value (get-safe-wme 'allocations-1) (get-
wme-name metropolis)))
         (reserve-GBIs (get-slot-value (get-safe-wme 'allocations-1) 'follow-
on-metropolis))
         (current-num-saved (calc-pop-saved metropolis-pop current-GBIs))
         (projected-num-saved (calc-pop-saved metropolis-pop (+ current-GBIs
1)))
```

```
        (follow-on-current-num-saved (calc-follow-on-pop-saved follow-on-pop
reserve-GBIs))
        (follow-on-projected-num-saved (calc-follow-on-pop-saved follow-on-
pop (- reserve-GBIs 1))))
    ;;(format t "~%metropolis pop: ~s" metropolis-pop)
    ;;(format t "~%Current GBIs: ~s" current-GBIs)
    (+ (- projected-num-saved current-num-saved)
       (- follow-on-projected-num-saved follow-on-current-num-saved))
    )
  )


(defun Allocate-GBI (metropolis-name)
  (let* ((allocations-wme (get-safe-wme 'allocations-1))
         (current-GBIs (get-slot-value allocations-wme metropolis-name))
         (reserve-GBIs (get-slot-value allocations-wme 'follow-on-
metropolis))
         (target-slot (get-slot metropolis-name (wme-type-slots (wme-type
allocations-wme))))
         (reserve-slot (get-slot 'follow-on-metropolis (wme-type-slots (wme-
type allocations-wme))))
         )
    (set-slot-value allocations-wme (slot-index target-slot) (+ current-GBIs
1))
    (set-slot-value allocations-wme (slot-index reserve-slot) (- reserve-GBIs
1))
    )
  )


;; load act-r
(clear-all)
(sgp :er t)

(chunk-type populations metropolis-1 metropolis-2 metropolis-3 metropolis-4
metropolis-5 follow-on-metropolis)
(chunk-type allocations metropolis-1 metropolis-2 metropolis-3 metropolis-4
metropolis-5 follow-on-metropolis)
(chunk-type gains expected-saved-1 expected-saved-2 expected-saved-3
expected-saved-4 expected-saved-5)
(chunk-type decision best-gain best-metropolis comparison pop-saved
comparison-gained-saved)
(chunk-type calculate-gain current-saved future-saved population GBIs gained-
saved)

(add-dm
 (scenario-1 isa populations
             metropolis-1 1839449
             metropolis-2 979044
             metropolis-3 1739986
             metropolis-4 978512
             metropolis-5 267409
             follow-on-metropolis 1260533)
 (allocations-1 isa allocations
                metropolis-1 0
                metropolis-2 0
                metropolis-3 0
```

```
                metropolis-4 0
                metropolis-5 0
                follow-on-metropolis 10)
 (gains-1 isa gains)
 (decision-1 isa decision)
 (test-calc-gain isa calculate-gain population 1000000 GBIs 1)
 )

(goal-focus decision-1)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
;; start the comparison process
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;

(p start-problem
   =goal>
   isa decision
   pop-saved nil
   !eval! (null *retrieval*)
   ==>
   !bind! =initial-pop-saved (calc-initial-saved)
   +retrieval>
   isa gains
   =goal>
   pop-saved =initial-pop-saved
   )


(p retrieve-gains
   =goal>
   isa decision
   - pop-saved nil
   !eval! (null *retrieval*)
   ==>
   +retrieval>
   isa gains
   )


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
;; do comparison for metropolis 1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;

(p calculate-metropolis-1-gains
   =goal>
   isa decision
   comparison nil
   =retrieval>
   isa gains
   expected-saved-1 nil
   ==>
   !bind! =expected-saved (calc-expected-saved 'metropolis-1)
   =retrieval>
```

```
   expected-saved-1 =expected-saved
   =goal>
   comparison 1)


;; compare metropolis-1
(p metropolis-1-is-best
   =goal>
   isa decision
   comparison 1
   best-gain =best-gain
   =retrieval>
   isa gains
   expected-saved-1 =es1
   >= expected-saved-1 =best-gain
   ==>
   =goal>
   best-gain =es1
   best-metropolis metropolis-1
   comparison nil
   )


;; compare metropolis-1 to best previous gain
(p metropolis-1-is-not-best
   =goal>
   isa decision
   comparison 1
   best-gain =best-gain
   =retrieval>
   isa gains
   expected-saved-1 =es1
   < expected-saved-1 =best-gain
   ==>
   =goal>
   comparison nil
   )


;; metropolis-1 is first in this decision cycle, so best by default
(p metropolis-1-is-better-than-nothing
   =goal>
   isa decision
   comparison 1
   best-gain nil
   =retrieval>
   isa gains
   expected-saved-1 =es1
   ==>
   =goal>
   best-gain =es1
   best-metropolis metropolis-1
   comparison nil
   )
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
;; do comparison for metropolis 2
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
(p calculate-metropolis-2-gains
   =goal>
   isa decision
   comparison nil
   =retrieval>
   isa gains
   expected-saved-2 nil
   ==>
   !bind! =expected-saved (calc-expected-saved 'metropolis-2)
   =retrieval>
   expected-saved-2 =expected-saved
   =goal>
   comparison 2)


;; compare metropolis-2 to best previous gain
(p metropolis-2-is-best
   =goal>
   isa decision
   comparison 2
   best-gain =best-gain
   =retrieval>
   isa gains
   expected-saved-2 =es2
   >= expected-saved-2 =best-gain
   ==>
   =goal>
   best-gain =es2
   best-metropolis metropolis-2
   comparison nil
   )


;; compare metropolis-2 to best previous gain
(p metropolis-2-is-not-best
   =goal>
   isa decision
   comparison 2
   best-gain =best-gain
   =retrieval>
   isa gains
   expected-saved-2 =es2
   < expected-saved-2 =best-gain
   ==>
   =goal>
   comparison nil
   )


;; metropolis-2 is first in this decision cycle, so best by default
(p metropolis-2-is-better-than-nothing
```

```
   =goal>
   isa decision
   comparison 2
   best-gain nil
   =retrieval>
   isa gains
   expected-saved-2 =es2
   ==>
   =goal>
   best-gain =es2
   best-metropolis metropolis-2
   comparison nil
   )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
;; do comparison for metropolis 3
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
(p calculate-metropolis-3-gains
   =goal>
   isa decision
   comparison nil
   =retrieval>
   isa gains
   expected-saved-3 nil
   ==>
   !bind! =expected-saved (calc-expected-saved 'metropolis-3)
   =retrieval>
   expected-saved-3 =expected-saved
   =goal>
   comparison 3)


;; compare metropolis-3 to best previous gain
(p metropolis-3-is-best
   =goal>
   isa decision
   comparison 3
   best-gain =best-gain
   =retrieval>
   isa gains
   expected-saved-3 =es3
   >= expected-saved-3 =best-gain
   ==>
   =goal>
   best-gain =es3
   best-metropolis metropolis-3
   comparison nil)


;; compare metropolis-3 to best previous gain
(p metropolis-3-is-not-best
   =goal>
   isa decision
   comparison 3
   best-gain =best-gain
```

```
   =retrieval>
   isa gains
   expected-saved-3 =es3
   < expected-saved-3 =best-gain
   ==>
   =goal>
   comparison nil)


;; metropolis-3 is first in this decision cycle, so best by default
(p metropolis-3-is-better-than-nothing
   =goal>
   isa decision
   comparison 3
   best-gain nil
   =retrieval>
   isa gains
   expected-saved-3 =es3
   ==>
   =goal>
   best-gain =es3
   best-metropolis metropolis-3
   comparison nil
   )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
;; do comparison for metropolis 4
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
(p calculate-metropolis-4-gains
   =goal>
   isa decision
   comparison nil
   =retrieval>
   isa gains
   expected-saved-4 nil
   ==>
   !bind! =expected-saved (calc-expected-saved 'metropolis-4)
   =retrieval>
   expected-saved-4 =expected-saved
   =goal>
   comparison 4)


;; compare metropolis-4 to best previous gain
(p metropolis-4-is-best
   =goal>
   isa decision
   comparison 4
   best-gain =best-gain
   =retrieval>
   isa gains
   expected-saved-4 =es4
   >= expected-saved-4 =best-gain
   ==>
   =goal>
```

```
   best-gain =es4
   best-metropolis metropolis-4
   comparison nil)


;; compare metropolis-4 to best previous gain
(p metropolis-4-is-not-best
   =goal>
   isa decision
   comparison 4
   best-gain =best-gain
   =retrieval>
   isa gains
   expected-saved-4 =es4
   < expected-saved-4 =best-gain
   ==>
   =goal>
   comparison nil)


;; metropolis-4 is first in this decision cycle, so best by default
(p metropolis-4-is-better-than-nothing
   =goal>
   isa decision
   comparison 4
   best-gain nil
   =retrieval>
   isa gains
   expected-saved-4 =es4
   ==>
   =goal>
   best-gain =es4
   best-metropolis metropolis-4
   comparison nil
   )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
;; do comparison for metropolis 5
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
(p calculate-metropolis-5-gains
   =goal>
   isa decision
   comparison nil
   =retrieval>
   isa gains
   expected-saved-5 nil
   ==>
   !bind! =expected-saved (calc-expected-saved 'metropolis-5)
   =retrieval>
   expected-saved-5 =expected-saved
   =goal>
   comparison 5)


;; compare metropolis-5 to best previous gain
```

```
(p metropolis-5-is-best
   =goal>
   isa decision
   comparison 5
   best-gain =best-gain
   =retrieval>
   isa gains
   expected-saved-5 =es5
   >= expected-saved-5 =best-gain
   ==>
   =goal>
   best-gain =es5
   best-metropolis metropolis-5
   comparison nil)


;; compare metropolis-5 to best previous gain
(p metropolis-5-is-not-best
   =goal>
   isa decision
   comparison 5
   best-gain =best-gain
   =retrieval>
   isa gains
   expected-saved-5 =es5
   < expected-saved-5 =best-gain
   ==>
   =goal>
   comparison nil)


;; metropolis-5 is first in this decision cycle, so best by default
(p metropolis-5-is-better-than-nothing
   =goal>
   isa decision
   comparison 5
   best-gain nil
   =retrieval>
   isa gains
   expected-saved-5 =es5
   ==>
   =goal>
   best-gain =es5
   best-metropolis metropolis-5
   comparison nil
   )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
(p allocate-GBI
   =goal>
   isa decision
   comparison nil
   best-metropolis =best-metropolis
   best-gain =best-gain
   >= best-gain 0
   pop-saved =pop-saved
```

```
   =retrieval>
   isa gains
   - expected-saved-1 nil
   - expected-saved-2 nil
   - expected-saved-3 nil
   - expected-saved-4 nil
   - expected-saved-5 nil
   ==>
   !output! "~%Choosing metropolis ~s for GBI allocation~%" =best-metropolis
   !eval! (Allocate-GBI =best-metropolis)
   !output! "~%Population saved: ~s" =best-gain
   !bind! =new-pop-saved (+ =best-gain =pop-saved)
   =retrieval>
   expected-saved-1 nil
   expected-saved-2 nil
   expected-saved-3 nil
   expected-saved-4 nil
   expected-saved-5 nil
   =goal>
   comparison nil
   best-gain nil
   best-metropolis nil
   pop-saved =new-pop-saved
   -retrieval>
)


(p No-GBI-allocation
   =goal>
   isa decision
   comparison nil
   best-metropolis =best-metropolis
   < best-gain 0
   =retrieval>
   isa gains
   expected-saved-1 =es1
   expected-saved-2 =es2
   expected-saved-3 =es3
   expected-saved-4 =es4
   expected-saved-5 =es5
   ==>
   !output! "~%No metropolis selected for GBI allocation (keeping
reserves)~%"
   =retrieval>
   expected-saved-1 nil
   expected-saved-2 nil
   expected-saved-3 nil
   expected-saved-4 nil
   expected-saved-5 nil
   =goal>
   comparison nil
   best-gain nil
   best-metropolis nil
   +retrieval>
   isa allocations)

(p State-Decision
```

```
   =goal>
   isa decision
   comparison nil
   pop-saved =pop-saved
   =retrieval>
   isa allocations
   metropolis-1 =met-1
   metropolis-2 =met-2
   metropolis-3 =met-3
   metropolis-4 =met-4
   metropolis-5 =met-5
   follow-on-metropolis =fom
   ==>
   !output! "~%Final allocations: City-1 ~s, City-2 ~s City-3 ~s, City-4 ~s,
City-5 ~s, Follow-on City ~s" =met-1 =met-2 =met-3 =met-4 =met-5 =fom
   !output! "Population saved: ~s~%" =pop-saved
   =goal>
   comparison nil
   -goal>
)
```

Appendix 3: Model Trace

```
dribbling to file "output.txt"

Loading Table C:\Documents and Settings\bbest\My Documents\Current
Projects\Optimal Training Systems\Simulations\Perceptual Noise 0p5 Expert
Feedback 0p5\scenarios.txt
 City1   99.326
    isa PERCEPT
    city 1
    pop 1839449
    miss 0
    area t
Modifying chunk CITY1 with pop 978332, and GBIs 0
 City2   100.124
    isa PERCEPT
    city 2
    pop 979044
    miss 0
    area t
Modifying chunk CITY2 with pop 344677, and GBIs 0
 City3   99.543
    isa PERCEPT
    city 3
    pop 1239986
    miss 0
    area t
Modifying chunk CITY3 with pop 250900, and GBIs 0
 City4   99.996
    isa PERCEPT
    city 4
    pop 78512
    miss 0
    area t
Modifying chunk CITY4 with pop 1821557, and GBIs 2
 City5   99.955
    isa PERCEPT
    city 5
    pop 46409
    miss 0
    area t
Modifying chunk CITY5 with pop 1728296, and GBIs 1
 Follow-On-City   99.301
    isa FOLLOW-ON
    prob 0.75
    pop 1260533
    miss 10
Modifying chunk FOLLOW-ON-CITY with pop 2368000, and GBIs 5
 Time  0.000: Next-City Selected
 Time  0.050: Next-City Fired
 Time  0.050: Info-City Selected
 Time  0.100: Info-City Fired
 Time  0.100: City1 Retrieved
 Time  0.100: Store-City Selected
 Time  0.150: Store-City Fired
 Time  0.150: Eval-Inc Selected
```

```
Time  0.200: Eval-Inc Fired
Time  1.200: Failure Retrieved
Time  1.200: Failure-Eval Selected
Time  1.250: Failure-Eval Fired
Time  1.250: Better-Eval Selected
Time  1.300: Better-Eval Fired
Time  1.300: Next-City Selected
Time  1.350: Next-City Fired
Time  1.350: Info-City Selected
Time  1.400: Info-City Fired
Time  1.400: City2 Retrieved
Time  1.400: Store-City Selected
Time  1.450: Store-City Fired
Time  1.450: Eval-Inc Selected
Time  1.500: Eval-Inc Fired
Time  2.500: Failure Retrieved
Time  2.500: Failure-Eval Selected
Time  2.550: Failure-Eval Fired
Time  2.550: Worse-Eval Selected
Time  2.600: Worse-Eval Fired
Time  2.600: Next-City Selected
Time  2.650: Next-City Fired
Time  2.650: Info-City Selected
Time  2.700: Info-City Fired
Time  2.700: City3 Retrieved
Time  2.700: Store-City Selected
Time  2.750: Store-City Fired
Time  2.750: Eval-Inc Selected
Time  2.800: Eval-Inc Fired
Time  3.800: Failure Retrieved
Time  3.800: Failure-Eval Selected
Time  3.850: Failure-Eval Fired
Time  3.850: Worse-Eval Selected
Time  3.900: Worse-Eval Fired
Time  3.900: Next-City Selected
Time  3.950: Next-City Fired
Time  3.950: Info-City Selected
Time  4.000: Info-City Fired
Time  4.000: City4 Retrieved
Time  4.000: Store-City Selected
Time  4.050: Store-City Fired
Time  4.050: Eval-Inc Selected
Time  4.100: Eval-Inc Fired
Time  5.100: Failure Retrieved
Time  5.100: Failure-Eval Selected
Time  5.150: Failure-Eval Fired
Time  5.150: Worse-Eval Selected
Time  5.200: Worse-Eval Fired
Time  5.200: Next-City Selected
Time  5.250: Next-City Fired
Time  5.250: Info-City Selected
Time  5.300: Info-City Fired
Time  5.300: City5 Retrieved
Time  5.300: Store-City Selected
Time  5.350: Store-City Fired
Time  5.350: Eval-Inc Selected
Time  5.400: Eval-Inc Fired
```

```
 Time  6.400: Failure Retrieved
 Time  6.400: Failure-Eval Selected
 Time  6.450: Failure-Eval Fired
 Time  6.450: Worse-Eval Selected
 Time  6.500: Worse-Eval Fired
 Time  6.500: Follow-On-City Selected
 Time  6.550: Follow-On-City Fired
 Time  6.550: Follow-On-City Retrieved
 Time  6.550: Eval-Follow-On Selected
 Time  6.600: Eval-Follow-On Fired
 Time  7.600: Failure Retrieved
 Time  7.600: Compute-Follow-On Selected
 Time  7.650: Compute-Follow-On Fired
 Time  7.650: Allocate Selected
Allocating GBI to City 1
Optimal move: Allocate GBI to City 1
 Time  7.700: Allocate Fired
 Time  7.700: Next-City Selected
 Time  7.750: Next-City Fired
 Time  7.750: Info-City Selected
 Time  7.800: Info-City Fired
 Time  7.800: City1 Retrieved
 Time  7.800: Store-City Selected
 Time  7.850: Store-City Fired
 Time  7.850: Eval-Inc Selected
 Time  7.900: Eval-Inc Fired
 Time  8.900: Failure Retrieved
 Time  8.900: Failure-Eval Selected
 Time  8.950: Failure-Eval Fired
 Time  8.950: Better-Eval Selected
 Time  9.000: Better-Eval Fired
 Time  9.000: Next-City Selected
 Time  9.050: Next-City Fired
 Time  9.050: Info-City Selected
 Time  9.100: Info-City Fired
 Time  9.100: City2 Retrieved
 Time  9.100: Store-City Selected
 Time  9.150: Store-City Fired
 Time  9.150: Eval-Inc Selected
 Time  9.200: Eval-Inc Fired
 Time 10.200: Failure Retrieved
 Time 10.200: Failure-Eval Selected
 Time 10.250: Failure-Eval Fired
 Time 10.250: Better-Eval Selected
 Time 10.300: Better-Eval Fired
 Time 10.300: Next-City Selected
 Time 10.350: Next-City Fired
 Time 10.350: Info-City Selected
 Time 10.400: Info-City Fired
 Time 10.400: City4 Retrieved
 Time 10.400: Wrong-City Selected
 Time 10.450: Wrong-City Fired
 Time 10.450: City3 Retrieved
 Time 10.450: Store-City Selected
 Time 10.500: Store-City Fired
 Time 10.500: Eval-Inc Selected
 Time 10.550: Eval-Inc Fired
```

```
 Time 11.550: Failure Retrieved
 Time 11.550: Failure-Eval Selected
 Time 11.600: Failure-Eval Fired
 Time 11.600: Worse-Eval Selected
 Time 11.650: Worse-Eval Fired
 Time 11.650: Next-City Selected
 Time 11.700: Next-City Fired
 Time 11.700: Info-City Selected
 Time 11.750: Info-City Fired
 Time 11.750: City4 Retrieved
 Time 11.750: Store-City Selected
 Time 11.800: Store-City Fired
 Time 11.800: Eval-Inc Selected
 Time 11.850: Eval-Inc Fired
 Time 12.850: Failure Retrieved
 Time 12.850: Failure-Eval Selected
 Time 12.900: Failure-Eval Fired
 Time 12.900: Worse-Eval Selected
 Time 12.950: Worse-Eval Fired
 Time 12.950: Next-City Selected
 Time 13.000: Next-City Fired
 Time 13.000: Info-City Selected
 Time 13.050: Info-City Fired
 Time 13.050: City5 Retrieved
 Time 13.050: Store-City Selected
 Time 13.100: Store-City Fired
 Time 13.100: Eval-Inc Selected
 Time 13.150: Eval-Inc Fired
 Time 14.150: Failure Retrieved
 Time 14.150: Failure-Eval Selected
 Time 14.200: Failure-Eval Fired
 Time 14.200: Worse-Eval Selected
 Time 14.250: Worse-Eval Fired
 Time 14.250: Follow-On-City Selected
 Time 14.300: Follow-On-City Fired
 Time 14.300: Follow-On-City Retrieved
 Time 14.300: Eval-Follow-On Selected
 Time 14.350: Eval-Follow-On Fired
 Time 15.350: Failure Retrieved
 Time 15.350: Compute-Follow-On Selected
 Time 15.400: Compute-Follow-On Fired
 Time 15.400: Allocate Selected
Allocating GBI to City 2
Optimal move: Allocate GBI to City 5
 Time 15.450: Allocate Fired
 Time 15.450: Next-City Selected
 Time 15.500: Next-City Fired
 Time 15.500: Info-City Selected
 Time 15.550: Info-City Fired
 Time 15.550: City1 Retrieved
 Time 15.550: Store-City Selected
 Time 15.600: Store-City Fired
 Time 15.600: Eval-Inc Selected
 Time 15.650: Eval-Inc Fired
 Time 16.650: Failure Retrieved
 Time 16.650: Failure-Eval Selected
 Time 16.700: Failure-Eval Fired
```

```
Time 16.700: Better-Eval Selected
Time 16.750: Better-Eval Fired
Time 16.750: Next-City Selected
Time 16.800: Next-City Fired
Time 16.800: Info-City Selected
Time 16.850: Info-City Fired
Time 16.850: City2 Retrieved
Time 16.850: Store-City Selected
Time 16.900: Store-City Fired
Time 16.900: Eval-Inc Selected
Time 16.950: Eval-Inc Fired
Time 17.950: Failure Retrieved
Time 17.950: Failure-Eval Selected
Time 18.000: Failure-Eval Fired
Time 18.000: Worse-Eval Selected
Time 18.050: Worse-Eval Fired
Time 18.050: Next-City Selected
Time 18.100: Next-City Fired
Time 18.100: Info-City Selected
Time 18.150: Info-City Fired
Time 18.150: City3 Retrieved
Time 18.150: Store-City Selected
Time 18.200: Store-City Fired
Time 18.200: Eval-Inc Selected
Time 18.250: Eval-Inc Fired
Time 19.250: Failure Retrieved
Time 19.250: Failure-Eval Selected
Time 19.300: Failure-Eval Fired
Time 19.300: Better-Eval Selected
Time 19.350: Better-Eval Fired
Time 19.350: Next-City Selected
Time 19.400: Next-City Fired
Time 19.400: Info-City Selected
Time 19.450: Info-City Fired
Time 19.450: City4 Retrieved
Time 19.450: Store-City Selected
Time 19.500: Store-City Fired
Time 19.500: Eval-Inc Selected
Time 19.550: Eval-Inc Fired
Time 20.550: Failure Retrieved
Time 20.550: Failure-Eval Selected
Time 20.600: Failure-Eval Fired
Time 20.600: Worse-Eval Selected
Time 20.650: Worse-Eval Fired
Time 20.650: Next-City Selected
Time 20.700: Next-City Fired
Time 20.700: Info-City Selected
Time 20.750: Info-City Fired
Time 20.750: City5 Retrieved
Time 20.750: Store-City Selected
Time 20.800: Store-City Fired
Time 20.800: Eval-Inc Selected
Time 20.850: Eval-Inc Fired
Time 21.850: Failure Retrieved
Time 21.850: Failure-Eval Selected
Time 21.900: Failure-Eval Fired
Time 21.900: Better-Eval Selected
```

```
 Time 21.950: Better-Eval Fired
 Time 21.950: Follow-On-City Selected
 Time 22.000: Follow-On-City Fired
 Time 22.000: Follow-On-City Retrieved
 Time 22.000: Eval-Follow-On Selected
 Time 22.050: Eval-Follow-On Fired
 Time 23.050: Failure Retrieved
 Time 23.050: Compute-Follow-On Selected
 Time 23.100: Compute-Follow-On Fired
 Time 23.100: Allocate Selected
Allocating GBI to City 5
Optimal move: Allocate GBI to City 5
 Time 23.150: Allocate Fired
 Time 23.150: Next-City Selected
 Time 23.200: Next-City Fired
 Time 23.200: Info-City Selected
 Time 23.250: Info-City Fired
 Time 23.250: City1 Retrieved
 Time 23.250: Store-City Selected
 Time 23.300: Store-City Fired
 Time 23.300: Eval-Inc Selected
 Time 23.350: Eval-Inc Fired
 Time 24.350: Failure Retrieved
 Time 24.350: Failure-Eval Selected
 Time 24.400: Failure-Eval Fired
 Time 24.400: Better-Eval Selected
 Time 24.450: Better-Eval Fired
 Time 24.450: Next-City Selected
 Time 24.500: Next-City Fired
 Time 24.500: Info-City Selected
 Time 24.550: Info-City Fired
 Time 24.550: City2 Retrieved
 Time 24.550: Store-City Selected
 Time 24.600: Store-City Fired
 Time 24.600: Eval-Inc Selected
 Time 24.650: Eval-Inc Fired
 Time 25.650: Failure Retrieved
 Time 25.650: Failure-Eval Selected
 Time 25.700: Failure-Eval Fired
 Time 25.700: Worse-Eval Selected
 Time 25.750: Worse-Eval Fired
 Time 25.750: Next-City Selected
 Time 25.800: Next-City Fired
 Time 25.800: Info-City Selected
 Time 25.850: Info-City Fired
 Time 25.850: City3 Retrieved
 Time 25.850: Store-City Selected
 Time 25.900: Store-City Fired
 Time 25.900: Eval-Inc Selected
 Time 25.950: Eval-Inc Fired
 Time 26.950: Failure Retrieved
 Time 26.950: Failure-Eval Selected
 Time 27.000: Failure-Eval Fired
 Time 27.000: Worse-Eval Selected
 Time 27.050: Worse-Eval Fired
 Time 27.050: Next-City Selected
 Time 27.100: Next-City Fired
```

```
 Time 27.100: Info-City Selected
 Time 27.150: Info-City Fired
 Time 27.150: City1 Retrieved
 Time 27.150: Wrong-City Selected
 Time 27.200: Wrong-City Fired
 Time 27.200: City4 Retrieved
 Time 27.200: Store-City Selected
 Time 27.250: Store-City Fired
 Time 27.250: Eval-Inc Selected
 Time 27.300: Eval-Inc Fired
 Time 28.300: Failure Retrieved
 Time 28.300: Failure-Eval Selected
 Time 28.350: Failure-Eval Fired
 Time 28.350: Worse-Eval Selected
 Time 28.400: Worse-Eval Fired
 Time 28.400: Next-City Selected
 Time 28.450: Next-City Fired
 Time 28.450: Info-City Selected
 Time 28.500: Info-City Fired
 Time 28.500: City5 Retrieved
 Time 28.500: Store-City Selected
 Time 28.550: Store-City Fired
 Time 28.550: Eval-Inc Selected
 Time 28.600: Eval-Inc Fired
 Time 29.600: Failure Retrieved
 Time 29.600: Failure-Eval Selected
 Time 29.650: Failure-Eval Fired
 Time 29.650: Worse-Eval Selected
 Time 29.700: Worse-Eval Fired
 Time 29.700: Follow-On-City Selected
 Time 29.750: Follow-On-City Fired
 Time 29.750: Follow-On-City Retrieved
 Time 29.750: Eval-Follow-On Selected
 Time 29.800: Eval-Follow-On Fired
 Time 30.800: Failure Retrieved
 Time 30.800: Compute-Follow-On Selected
 Time 30.850: Compute-Follow-On Fired
 Time 30.850: Allocate Selected
Allocating GBI to City 1
Optimal move: Allocate GBI to City 1
 Time 30.900: Allocate Fired
 Time 30.900: Next-City Selected
 Time 30.950: Next-City Fired
 Time 30.950: Info-City Selected
 Time 31.000: Info-City Fired
 Time 31.000: City1 Retrieved
 Time 31.000: Store-City Selected
 Time 31.050: Store-City Fired
 Time 31.050: Eval-Inc Selected
 Time 31.100: Eval-Inc Fired
 Time 32.100: Failure Retrieved
 Time 32.100: Failure-Eval Selected
 Time 32.150: Failure-Eval Fired
 Time 32.150: Better-Eval Selected
 Time 32.200: Better-Eval Fired
 Time 32.200: Next-City Selected
 Time 32.250: Next-City Fired
```

```
Time 32.250: Info-City Selected
Time 32.300: Info-City Fired
Time 32.300: City2 Retrieved
Time 32.300: Store-City Selected
Time 32.350: Store-City Fired
Time 32.350: Eval-Inc Selected
Time 32.400: Eval-Inc Fired
Time 33.400: Failure Retrieved
Time 33.400: Failure-Eval Selected
Time 33.450: Failure-Eval Fired
Time 33.450: Worse-Eval Selected
Time 33.500: Worse-Eval Fired
Time 33.500: Next-City Selected
Time 33.550: Next-City Fired
Time 33.550: Info-City Selected
Time 33.600: Info-City Fired
Time 33.600: City3 Retrieved
Time 33.600: Store-City Selected
Time 33.650: Store-City Fired
Time 33.650: Eval-Inc Selected
Time 33.700: Eval-Inc Fired
Time 34.700: Failure Retrieved
Time 34.700: Failure-Eval Selected
Time 34.750: Failure-Eval Fired
Time 34.750: Better-Eval Selected
Time 34.800: Better-Eval Fired
Time 34.800: Next-City Selected
Time 34.850: Next-City Fired
Time 34.850: Info-City Selected
Time 34.900: Info-City Fired
Time 34.900: City4 Retrieved
Time 34.900: Store-City Selected
Time 34.950: Store-City Fired
Time 34.950: Eval-Inc Selected
Time 35.000: Eval-Inc Fired
Time 36.000: Failure Retrieved
Time 36.000: Failure-Eval Selected
Time 36.050: Failure-Eval Fired
Time 36.050: Better-Eval Selected
Time 36.100: Better-Eval Fired
Time 36.100: Next-City Selected
Time 36.150: Next-City Fired
Time 36.150: Info-City Selected
Time 36.200: Info-City Fired
Time 36.200: City5 Retrieved
Time 36.200: Store-City Selected
Time 36.250: Store-City Fired
Time 36.250: Eval-Inc Selected
Time 36.300: Eval-Inc Fired
Time 37.300: Failure Retrieved
Time 37.300: Failure-Eval Selected
Time 37.350: Failure-Eval Fired
Time 37.350: Better-Eval Selected
Time 37.400: Better-Eval Fired
Time 37.400: Follow-On-City Selected
Time 37.450: Follow-On-City Fired
Time 37.450: Follow-On-City Retrieved
```

```
 Time 37.450: Eval-Follow-On Selected
 Time 37.500: Eval-Follow-On Fired
 Time 38.500: Failure Retrieved
 Time 38.500: Compute-Follow-On Selected
 Time 38.550: Compute-Follow-On Fired
 Time 38.550: Allocate Selected
Allocating GBI to City 5
Optimal move: no allocation (keep reserve level)
 Time 38.600: Allocate Fired
 Time 38.600: Next-City Selected
 Time 38.650: Next-City Fired
 Time 38.650: Info-City Selected
 Time 38.700: Info-City Fired
 Time 38.700: City1 Retrieved
 Time 38.700: Store-City Selected
 Time 38.750: Store-City Fired
 Time 38.750: Eval-Inc Selected
 Time 38.800: Eval-Inc Fired
 Time 39.800: Failure Retrieved
 Time 39.800: Failure-Eval Selected
 Time 39.850: Failure-Eval Fired
 Time 39.850: Better-Eval Selected
 Time 39.900: Better-Eval Fired
 Time 39.900: Next-City Selected
 Time 39.950: Next-City Fired
 Time 39.950: Info-City Selected
 Time 40.000: Info-City Fired
 Time 40.000: City2 Retrieved
 Time 40.000: Store-City Selected
 Time 40.050: Store-City Fired
 Time 40.050: Eval-Inc Selected
 Time 40.100: Eval-Inc Fired
 Time 41.100: Failure Retrieved
 Time 41.100: Failure-Eval Selected
 Time 41.150: Failure-Eval Fired
 Time 41.150: Worse-Eval Selected
 Time 41.200: Worse-Eval Fired
 Time 41.200: Next-City Selected
 Time 41.250: Next-City Fired
 Time 41.250: Info-City Selected
 Time 41.300: Info-City Fired
 Time 41.300: City3 Retrieved
 Time 41.300: Store-City Selected
 Time 41.350: Store-City Fired
 Time 41.350: Eval-Inc Selected
 Time 41.400: Eval-Inc Fired
 Time 42.400: Failure Retrieved
 Time 42.400: Failure-Eval Selected
 Time 42.450: Failure-Eval Fired
 Time 42.450: Better-Eval Selected
 Time 42.500: Better-Eval Fired
 Time 42.500: Next-City Selected
 Time 42.550: Next-City Fired
 Time 42.550: Info-City Selected
 Time 42.600: Info-City Fired
 Time 42.600: City4 Retrieved
 Time 42.600: Store-City Selected
```

```
 Time 42.650: Store-City Fired
 Time 42.650: Eval-Inc Selected
 Time 42.700: Eval-Inc Fired
 Time 43.700: Failure Retrieved
 Time 43.700: Failure-Eval Selected
 Time 43.750: Failure-Eval Fired
 Time 43.750: Worse-Eval Selected
 Time 43.800: Worse-Eval Fired
 Time 43.800: Next-City Selected
 Time 43.850: Next-City Fired
 Time 43.850: Info-City Selected
 Time 43.900: Info-City Fired
 Time 43.900: City5 Retrieved
 Time 43.900: Store-City Selected
 Time 43.950: Store-City Fired
 Time 43.950: Eval-Inc Selected
 Time 44.000: Eval-Inc Fired
 Time 45.000: Failure Retrieved
 Time 45.000: Failure-Eval Selected
 Time 45.050: Failure-Eval Fired
 Time 45.050: Worse-Eval Selected
 Time 45.100: Worse-Eval Fired
 Time 45.100: Follow-On-City Selected
 Time 45.150: Follow-On-City Fired
 Time 45.150: Follow-On-City Retrieved
 Time 45.150: No-More-Missiles Selected
 Time 45.200: No-More-Missiles Fired
 Time 45.200: No-Allocate Selected
Desired missiles for city 1 population 978332 missiles allocated 2: 2
Desired missiles for city 2 population 344677 missiles allocated 1: 1
Desired missiles for city 3 population 250900 missiles allocated 0: 0
Desired missiles for city 4 population 1821557 missiles allocated 2: 2
Desired missiles for city 5 population 1728296 missiles allocated 3: 2
Desired missiles for follow-on city 6 population 2368000 probability   0.25
missiles left 0: 1
 Time 45.250: No-Allocate Fired
 Time 45.250: Next-Feedback Selected
 Time 45.300: Next-Feedback Fired
 Time 45.300: Feedback-City Selected
 Time 45.350: Feedback-City Fired
 Time 45.715: Feedback5 Retrieved
 Time 45.715: Store-Feedback Selected
 Time 45.765: Store-Feedback Fired
 Time 45.765: Right-Allocation Selected
 Time 45.815: Right-Allocation Fired
 Time 45.815: Next-Feedback Selected
 Time 45.865: Next-Feedback Fired
 Time 45.865: Feedback-City Selected
 Time 45.915: Feedback-City Fired
 Time 46.047: Feedback6 Retrieved
 Time 46.047: Store-Feedback Selected
 Time 46.097: Store-Feedback Fired
 Time 46.097: Right-Allocation Selected
 Time 46.147: Right-Allocation Fired
 Time 46.147: Next-Feedback Selected
 Time 46.197: Next-Feedback Fired
 Time 46.197: Feedback-City Selected
```

```
Time 46.247: Feedback-City Fired
Time 46.451: Feedback7 Retrieved
Time 46.451: Store-Feedback Selected
Time 46.501: Store-Feedback Fired
Time 46.501: Right-Allocation Selected
Time 46.551: Right-Allocation Fired
Time 46.551: Next-Feedback Selected
Time 46.601: Next-Feedback Fired
Time 46.601: Feedback-City Selected
Time 46.651: Feedback-City Fired
Time 47.039: Feedback8 Retrieved
Time 47.039: Store-Feedback Selected
Time 47.089: Store-Feedback Fired
Time 47.089: Right-Allocation Selected
Time 47.139: Right-Allocation Fired
Time 47.139: Next-Feedback Selected
Time 47.189: Next-Feedback Fired
Time 47.189: Feedback-City Selected
Time 47.239: Feedback-City Fired
Time 47.521: Feedback9 Retrieved
Time 47.521: Store-Feedback Selected
Time 47.571: Store-Feedback Fired
Time 47.571: Less-Allocation Selected
Time 47.621: Less-Allocation Fired
Time 47.621: Next-Feedback Selected
Time 47.671: Next-Feedback Fired
Time 47.671: Feedback-City Selected
Time 47.721: Feedback-City Fired
Time 47.983: Feedback10 Retrieved
Time 47.983: Store-Follow-On Selected
Time 48.033: Store-Follow-On Fired
Time 48.033: More-Reserve Selected
Time 48.083: More-Reserve Fired
Time 48.083: End-Feedback Selected
Time 48.133: End-Feedback Fired
```

Appendix 4: Experiment Instructions

- INSTRUCTIONS FOR ALL CONDITIONS: The Power Point you saw described how to add and remove missiles interceptors, but it didn't cover one important detail. Any time you want to save a certain distribution and move on, you have to hit the Reallocate button. If you want to allocate missiles to one city, and then allocate missiles somewhere else, you must first hit the Reallocate button before you move on. So essentially, this button is like a 'Save' button which registers your decision. The same is true when you want to remove missiles. If you don't hit this button, then anything you add or subtract will be erased. Also, if you make any last-second changes, remember to hit the button before time expires or it won't be saved and submitted. Moving on, in terms of the information you have to consider, keep in mind that besides the 5 cities presented here, there is another element involved. You also have to consider the possibility of a future attack on another city. Any missile interceptors that you do not distribute will then be reserved for the follow-on attack. So keep in mind that you have about 40 seconds to allocate these resources, and that your goal ultimately is maximize the number of people saved. Good luck.

- ADDENDUM, FOR FEEDBACK CONDITION: After each trial, you'll see a feedback window. This screen describes what happened in this particular situation, like whether any cities were attacked. It's also presents a comparison between your allocation of missile interceptors and the optimal allocation. Lastly, there is a display of how many people would be expected to be saved with your arrangement versus the optimal one.

- EXTRA ADDENDUM, FOR FEEDBACK BEEPING CONDITION: Along with the feedback window, you will also be given another type of feedback, a more direct type. The beeping is a cue that activates when you did not make the optimal decision for that particular step. When you don't hear the beep, it means that you chose the most optimal choice for that specific step.